КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ТАРАСА ШЕВЧЕНКА
Факультет комп'ютерних наук та кібернетики
Кафедра математичної інформатики

КВАЛІФІКАЦІЙНА РОБОТА
на здобуття ступеня магістра

за спеціальністю 122 Комп'ютерні науки

на тему:

РОЗРОБКА ТА АВТОМАТИЗАЦІЯ СИСТЕМИ ПАРКС НА ПЛАТФОРМІ .NET

Виконав студент 2-го курсу магістратури
Олексій БОГУСЕВИЧ

(підпис)

Науковий керівник:
асистент, кандидат технічних наук
Олексій ФЕДОРУС

(підпис)

Засвідчую, що в цій роботі немає запозичень з праць інших авторів без відповідних посилань.

Студент

(підпис)

Роботу розглянуто й допущено до захисту на засіданні кафедри математичної інформатики
«___» _____20__р.,
протокол № ___

Завідувач кафедри
В. М. Терещенко _____
(підпис)

Київ – 2023

РЕФЕРАТ

Робота складається зі вступу, 3 розділів, висновків, списку використаних джерел (40 найменувань). Робота містить 18 рисунків, 3 таблиці. Загальний обсяг становить 60 сторінок, основний текст роботи викладено на 46 сторінках.

Ключові слова: АВТОМАТИЗАЦІЯ, БАЗИ ДАНИХ, ВЕБ-РОЗРОБКА, КОНТЕЙНЕРИ, МЕРЕЖЕВА КОМУНІКАЦІЯ, ОРКЕСТРАЦІЯ, ПАРАЛЕЛЬНІ ОБЧИСЛЕННЯ, РЕКУРСИВНІ ОБЧИСЛЕННЯ, ХМАРНІ ОБЧИСЛЕННЯ

Об’єктом роботи є система ПАРКС, що була розроблена провідними українськими вченими на факультеті комп’ютерних наук та кібернетики Київського національного університету імені Тараса Шевченка. Зокрема, її реалізація для платформи .NET.

Метою роботи є модернізація та автоматизація компонент системи шляхом аналізу та визначення особливостей попередніх напрацювань в сфері, оновленням цільової платформи до найсучаснішої збірки, позбавленням реалізації залежностей від конкретної операційної оболонки, створенням графічного інтерфейсу для забезпечення інтуїтивної взаємодії, а також налаштуванням та стабілізацією автоматичного розгортання необхідної інфраструктури.

Методами дослідження та розроблення є опанування дотичних праць, ознайомлення з запропонованими в них реалізаціями, робота з технічною літературою на тему, проектування та відлагодження контейнеризованих застосунків, налаштування мережевої комунікації, проектування веб інтерфейсів, побудова рішень для оркестрації контейнерів, створення скриптів розгортання.

Інструментами розроблення є вільно поширювані інтегровані середовища Visual Studio Code та Visual Studio Professional, система контролю версій Git, додаток для роботи з контейнерами Docker Desktop, публічний реєстр образів Docker Hub, локальний менеджер Kubernetes minikube, хмарна платформа Microsoft Azure, інтерфейс командного рядку Azure CLI, а також розширення

Azure Account, ARM Tools, ARM Template Viewer, Bicep, Docker, Kubernetes, Bridge to Kubernetes та PlantUML для зазначених вище IDE.

Взаємозв’язок з іншими роботами: заснована на результатах [1] та [2], дана праця має на меті розвинути підняту в зазначених роботах тему виконання паралельних алгоритмів на комп’ютерній мережі для платформи .NET.

Результати роботи: керуючись принципами і засадами, закладеними в попередніх проектах, та взявши за основу найновішу версію платформи .NET, створено сучасну реалізацію ПАРКС, що складається з TCP-сервера у вигляді фонового сервісу для .NET Generic Host, програмного веб-інтерфейсу на базі ASP.NET Core Web API, графічного веб-інтерфейсу ASP.NET Core Blazor та спільного стійкого сховища; перераховані компоненти визначено в YAML-файлах публікації Kubernetes та docker-compose, обидва з яких є агностичними до хмарних та внутрішніх середовищ розгортання; окремо, за допомогою інтерпретатора Bicep, згенеровано ARM шаблон для автоматичної публікації усього рішення на керований сервіс AKS.

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАЧЕННЯ

- АМ – алгоритмічний модуль
- БД – база даних
- ВМ – віртуальна машина
- ЕОМ – електронно-обчислювальна машина
- КП – керуючий простір
- ОМ – обчислювальна машина
- ПАРКС – паралельні асинхронні рекурсивно керовані системи
- ПК – програмний канал
- ПС – паралельні системи
- AKS – Azure Kubernetes Service
- ARM – Azure Resource Management
- CLI – Command Line Interface
- DSL – Domain Specific Language
- IDE – Integrated Development Environment
- IPC – Inter-process Communication
- ITC – Inter-thread Communication

ВСТУП

Оцінка сучасного стану об’єкта розробки. Останні декілька десятиліть розвиток інформаційних технологій набрав значних обертів. Незважаючи на те, що на думку багатьох епоха закону Мура добігає кінця [3], приголомшуюча швидкість та безпомилкова точність розрахунків сучасних комп’ютерів невблаганно наближається до показників людського мозку, який і собі вже не проти делегувати частину власних повноважень своєму штучному протеже [4].

Водночас, на п’єдесталі рушіїв цього прогресу окреме місце виділено під паралельні обчислення, механізм яких дозволяє отримати ще суттєвіше прискорення, асимптотично пропорційне кількості доданих вузлів. З огляду ж на безперебійний зріст об’єму даних і аналогічно значущий попит на їх обробку, паралелізм стає необхідністю, а не опціонною надбудовою. Дійсно, паралельна обробка інформації має широкий спектр застосувань, зі складу якого можна, до прикладу, виділити космічні корабельні системи NASA, майнінг криптовалют, Інтернет речей тощо [5]. Вочевидь, ця невід’ємна складова технологічного простору не має зупинятись на досягнутому, а натомість мусить пропонувати все новіші, швидші, простіші та зручніші рішення, аби задовольняти все більш складні та вимогливі запити людства.

Розподілена система паралельних обчислень ПАРКС є однією з розробок в окресленій сфері, яка також потребує модернізації. На сьогоднішній день існує декілька версій ПАРКС для різних платформ: Java, Python та .NET. Остання була обрана цільовою в двох випадках: в оригінальному варіанті 2018 року [1] та для моделювання операцій реляційної алгебри вибору у 2021 [2]. Обидва проекти були побудовані на базі версій більше не підтримуваного та прив’язаного до операційної системи Windows фреймворку .NET Framework (4.6.1 та 4.5 відповідно). Окрім того, хоч процес розгортання і було дещо спрощено у [6] за рахунок контейнеризації та шаблонів для віртуальних машин, інструкції, що його пояснюють, залишились доволі трудомісткими, а інфраструктура – надлишковою.

Актуальність роботи та підстави для її виконання. Кількість наукових робіт присвячених системі ПАРКС в останні роки доводить актуальність технології. Окремо потрібно відзначити, що ПАРКС вивчається студентами ОС «Бакалавр» та «Магістр» на факультеті комп'ютерних наук та кібернетики КНУ імені Тараса Шевченка в рамках програм «Розподілене і паралельне програмування», «Системи паралельного програмування» та «Розподілені системи обробки інформації» [7]. Звідси, удосконалення системи має стати не лише внеском в розвиток перспективної платформи, а й поліпшенням досвіду здобувачів технічної освіти.

Підстави для виконання даної роботи були явно окреслені у вже наведених працях [1, 2]: «*Потенційним шляхом розвитку системи ПАРКС.NET є портування її на багатоплатформну платформу .NET Core, що дозволить використовувати систему ПАРКС.NET також на операційних системах Linux.*» [1], «*Вона прив'язана до технології WCF, яка не є мультиплатформенною… можливості її перенесення на інші платформи (разом з розвитком .Net Core…*" [2]. Поряд з тим, власний досвід розгортання компонент ПАРКС під час навчання на факультеті, а також численні відгуки одногрупників про складність налаштування системи саме для платформи .NET стали додатковими аргументами на користь рішень щодо спрощення користувацького інтерфейсу та акценту на автоматизації розгортання інфраструктури.

Мета й завдання роботи. Метою роботи є побудова системи ПАРКС на базі сучасної версії платформи .NET, а також автоматизація її розгортання.

Реалізація цієї мети передбачає виконання таких завдань:

1. проаналізувати попередні напрацювання в сфері та систематизувати існуючий досвід;
2. спроектувати та розробити аналог системи у термінах сучасних засобів програмування на платформі .NET;
3. провести тестування та переконатися у ефективності розробленої системи та її відповідності класичній концепції ПАРКС;

4. створити засоби розгортання системи, що вимагали б якомога меншого втручання людини в процес.

Об’єкт, методи й засоби розроблення. Об’єктом роботи є система ПАРКС. Завдання роботи розв’язувались за допомогою наступних методів і засобів.

Завдання 1:

- Аналіз наукових статей на тему
- Аналіз коду та локальне відлагодження реалізацій PARCS.NET [1] та PARCS-WCF [2]

Завдання 2:

- Плагін-орієнтовна архітектура
- Мережева комунікація з використанням TCP та HTTP/S
- REST API
- Рефлексія
- Платформа .NET 7 та мова програмування C# 10
- Фреймворк ASP.NET Core 7.0, а саме Web API та Blazor
- Менеджер пакетів NuGet
- Платформа Docker та рішення Docker Compose
- Спільні сховища Docker Volumes

Завдання 3:

- Мануальне тестування
- XUnit
- Benchmark.NET

Завдання 4:

- Система Kubernetes 1.25
- Керований сервіс Azure Kubernetes Service
- Мова програмування Bicep та шаблони Azure Resource Manager

РОЗДІЛ 1. АНАЛІЗ СУЧАСНОГО СТАНУ РОЗРОБКИ

1.1. ПАРКС-технологія програмування

ПАРКС-технологія програмування являє собою деяку множину програмних засобів, які забезпечують процес розробки і реалізації алгоритмів паралельної обробки інформації і базується на концепції керуючого простору (КП) [8].

Основними термінами ПАРКС-технології програмування є:

- Керуючий простір (КП);
- Точка;
- Програмний канал (ПК);
- Алгоритмічний модуль (АМ).

ПАРКС-технологія програмування базується на концепції керуючого простору (КП) – динамічного графу, за допомогою якого описується логічна й комунікаційна структура досліджуваної задачі (системи) і відображаються динамічні зміни в ній.

ПАРКС-технологія програмування дозволяє:

- підтримувати структурне паралельне програмування на основі статичного та динамічного паралелізму;
- використовувати різноманітні засоби опису рекурсії по даних і рекурсії по керуванню;
- на логічному рівні в явному вигляді описувати розподілення ресурсів, схеми комутації і перекомутації зв’язків або отримувати логічну структуру системи, розподілення ресурсів і схему перекомутації як результат обробки і побудови керуючого простору;
- будувати систему віртуальних ПАРКС-машин, що забезпечують можливість паралельної обробки інформації: ефективна реалізація системи віртуальних ПАРКС-машин можлива лише на широкому класі паралельних і спеціалізованих ЕОМ [9].

1.2. ПАРКС-моделі

За допомогою засобів технології ПАРКС можна створювати алгоритмічні моделі будь-якої складності. Такі моделі, отримані шляхом використання технології ПАРКС, називають ПАРКС-моделями. Побудова цих моделей відбувається на двох рівнях: логічному та програмному. На логічному рівні засоби ПАРКС-технології програмування дозволяють користувачу в явному вигляді описувати або в неявному отримувати розподіл ресурсів і всі можливі схеми комутації та перекомутації (з врахуванням рекурсивного розгортання алгоритму). На програмному рівні визначається управління, дані й правила взаємодії між даними та керуванням з урахуванням відповідної логічної структури.

ПАРКС-модель визначається наступним чином:

1. Виділяється деякий набір базових алгоритмічних перетворень інформації – алгоритмічних модулів;
2. В процесі функціонування моделі допускається (в тому числі рекурсивне) створення активних копій АМ, причому можливі «мутаційні» зміни АМ, що визначаються обставинами;
3. Процес функціонування моделі складається зі створення та знищення активних копій АМ, їх частково-децентралізованого асинхронного функціонування та динамічної взаємодії [10].

1.3. Віртуальні ПАРКС-машини

Розробка ПАРКС - моделей здійснюється на базі ієрархічної структури віртуальних ПАРКС - машин. Розглянемо програмні засоби й можливості, надані кожною віртуальною ПАРКС - машиною для розширення базової алгоритмічної мови.

Рівень 1. Програмні засоби для побудови й модифікації КП.

КП складається з точок і каналів, що з'єднують точки. У кожній точці КП на наступному рівні ієрархічного представлення може розміщуватися новий КП. При побудові й обробці КП припустиме використання рекурсії за структурою

КП. Кожна точка має свій тип, що визначає її властивості й алгоритм функціонування (монітор) для рівня 2. Наприклад, точка може моделювати логічний процесор, розподілюваний ресурс, блок пам'яті тощо. Точка КП однозначно ідентифікується деяким логічним номером. Канали КП призначені для забезпечення взаємозв'язків між точками. Вони також типізовані. За допомогою типів каналів задається, зокрема, ієрархічна структура КП. Для кожного алгоритму існує адекватна йому структура КП, що відображає його логічну структуру.

Програмні засоби рівня 1 повинні забезпечувати наступні можливості:

– створити нову точку КП заданого типу і визначити для неї новий логічний номер;

– з'єднати точки КП каналом заданого типу;

– визначити, які точки КП приєднані до заданого, і за допомогою яких каналів;

– знищити точку КП і всі канали, що з'єднують її з іншими точками;

– знищити канал заданого типу, що з'єднує точки КП;

– визначити логічний номер і тип точки КП.

Рівень 2. Програмні засоби для збереження й передачі інформації за допомогою КП.

Монітор точки КП, що відповідає її типу, визначає правила збереження, прийому і передачі інформації у точці КП. Крім того, монітор визначає її алгоритм функціонування. На рівні 2 у рамках каналів КП виділяються нумеровані канали, за допомогою яких уточняється структура взаємозв'язків між точками КП.

Програмні засоби рівня 2 повинні забезпечувати наступні можливості:

– створити скінченне число моніторів (на основі засобів базової алгоритмічної мови, мови асемблера тощо);

– створити блок інформації;

– записати в канал із заданим номером (можлива явна вказівка точки КП, до якої приєднаний даний канал) блок інформації з визначеним пріоритетом;

– перевірити, чи маються в каналі із заданим номером блоки інформації;

– прочитати з каналу із заданим номером черговий блок інформації;

– очікувати запису блоку інформації у кожній з каналів із заданими номерами.

Реалізація програмних засобів рівнів 1 і 2 залежать від архітектури машини і її операційної системи. Програмні засоби наступних рівнів і реалізація залежать від базової алгоритмічної мови.

Рівень 3. Програмні засоби, що забезпечують роботу з алгоритмічними модулями.

Поняття алгоритмічного модуля (АМ) призначено для уточнення одиниці паралельної роботи базової алгоритмічної мови. За допомогою АМ здійснюється побудова, модифікація і «наповнення» КП.

Опис АМ являє собою розширення опису відповідної конструкції базової мови операціями віртуальних рівнів 1-3. Найпростіший опис АМ — це дефініція відповідної конструкції базової мови.

Таким чином, програмні засоби рівня 3 повинні забезпечувати наступні можливості:

– створити, знищити й модифікувати опис АМ;

– створити активну копію АМ і приписати її для виконання в точку КП (логічний номер точки КП визначає динамічне ім'я активної копії АМ; планування виконання активних копій АМ у точках КП здійснюється відповідними моніторами);

– послати повідомлення активній копії АМ (відкрити доступ до розподілених структур даних);

– прийняти повідомлення від активної копії АМ (закрити доступ до розподілених структур даних);

– визначити головну активну копію АМ (із створення якої починається функціонування ПАРКС - моделі);

– здійснити вибіркове очікування й альтернативне виконання в тілі активної копії АМ.

Рекурсія на рівні АМ виявляється у тому, що активні копії АМ можуть створювати нові активні копії по описах АМ (у тому числі по власному зразку), а також нові описи АМ (у тому числі модифікувати власні описи).

Апарат створення, знищення, модифікації, приписування активних копій АМ у точки КП, взаємодії активних копій АМ, обробки даних як керування, а керування — як даних, дозволяє описувати рекурсію за даними, рекурсію по керуванню та їхній комбінації.

Рівень 4. Програмні засоби, що забезпечують процедурно - об'єктно - орієнтоване паралельне програмування.

Технологія процедурно та об'єктно-орієнтованого програмування визначається й обмежується базовою алгоритмічною мовою. Якщо взяти за основу об'єктно-орієнтовану Ада-технологію — механізм пакетів і задач, уніфікований виклик процедур пакетів і точок входів задач, поняття рандеву й інші, то на заданій базовій, алгоритмічній мові вона працює наступним чином.

Вибирається програмна конструкція базової мови, що відповідає по структурі поняттю пакета (задачі) у мові Ada. Наприклад, модуль - у мові Modula-2, файл, що містить опис сукупності функцій, — у мові С тощо. Обрана конструкція розглядається як опис АМ. Розходження між пакетами й задачами визначається на рівні КП при приписці активних копій АМ у точки різних типів (монітори їх задають відповідно правилу обробки звертань до пакетів і задач). Уніфікований виклик процедур і точок входів транслюється на рівень 3, де розписується за допомогою програмних засобів, що забезпечують взаємодію активних копій АМ.

На наступних рівнях розробляються програмні засоби, використання яких забезпечує автоматичну генерацію паралельних програм за їхніми специфікаціями [9].

1.4. Попередні реалізації системи

Від першої теоретичної праці Анатолія Васильовича Анісімова, де згадується абревіатура «ПАРКС», у 1982 році [11], і до сьогодення було написано та опубліковано чимало наукових статей на тему, а також створено так само багато реалізацій системи ПАРКС для різних платформ. До переліку останніх належать PARCS-PASCAL, PARCS-Modula2, PARCS-FORTRAN, PARCS-C, PARCS-Java, PARCS-Cuda, PARCS-Python, PARCS-NET та PARCS-WCF [2]. Найбільшого поширення набула PARCS-JAVA, на основі якої було написано навчальний посібник для студентів факультету комп'ютерних наук та кібернетики [9].

Оскільки дана робота має на меті зробити внесок в сімейство ПАРКС-систем платформи .NET, зупинимось детальніше саме на версіях PARCS-NET [1] та PARCS-WCF [2].

1.5. PARCS-NET

Спочатку розглянемо PARCS-NET. Це перше втілення ідей ПАРКС на базі .NET, яке й донині залишається взірцевим. Запропонована система складається з таких компонент:

- спільних бібліотек класів Parcs [1212] та Parcs.Module.CommandLine [13], опублікованих в якості NuGet-пакетів (збірок, що містять готовий до використання код для проектів .NET). Перша з них надає доступ до абстракцій, що необхідні для створення алгоритмічних модулів, друга – спрощений механізм передачі CLI параметрів.
- консольного застосунку DaemonPr, що є базовою обчислювальною одиницею для побудови і виконання ПАРКС-модулів, який в точці входу розгортає TCP-сервер і починає активно очікувати на нові підключення. Клієнти цього сервера мають комунікувати з ним за деяким узгодженим протоколом, що передбачає надсилання сигналів на початку кожного наступного повідомлення. Прикладами таких сигналів є ReceiveTask (ініціалізація задачі), ExecuteClass

(запуск на виконання конкретного класу), CancelJob (скасування задачі). Зрозуміло, кожен такий сигнал задає конкретний спосіб оброки запиту.

- консольного застосунку HostServer, що є коренем ієрархії КП, який на початку роботи так само запускає TCP-сервер із деякими вже іншими наперед заданими типами сигналів. До прикладів сигналів, які обробляються HostServer, належать PointCreated (запит на створення точки), PointDeleted (запит на знищення точки), IpAddress (запит на видачу IP-адреси сервера).
- веб-застосунків RestApi та HostServer.Web для моніторингу поточного стану виконання задач, перший з яких також дозволяє запускати алгоритмічні модулі. Серед визначених операцій: створення користувачів, отримання інформаційних реєстрів, виконання модулів, перелік доступних модулів, перелік обчислювальних машин та задач. Обидва API базуються на реалізації OWIN (Open Web Interface for .NET) – стандарту, що визначає спосіб взаємодії між веб-сервером та веб-додатком в середовищі .NET. В часи написання HostServer такі бібліотеки, як використана тут Microsoft.Owin, були одним з найгнучкіших рішень для побудови веб-інтерфейсів.

На рисунку 1 наведено базисний модуль обчислення інтегралу, приклад АМ в PARCS-NET. Цей модуль, як і всі інші тут, виконує контракт інтерфейсу IModule, визначеного в спільному пакеті [**Error! Reference source not found.**]. Згідно інтерфейсу, єдиний метод Run приймає два параметри на вхід: ModuleInfo, через який можна створювати нові точки, доступатись до батьківського каналу та поточного завдання, а також CancellationToken, вбудований в .NET механізм відміни асинхронних операцій.

Прямокутником з номером «1» виділено блок коду створення КП (топології мережі), прямокутником з номером «2» - розподіл вхідних даних між точками («наповнення» логічної структури КП), прямокутником з номером «3» - збір вихідних даних з точок (агрегація результатів обчислень). Як можна побачити, код модуля є доволі абстрактним і легким для сприйняття.

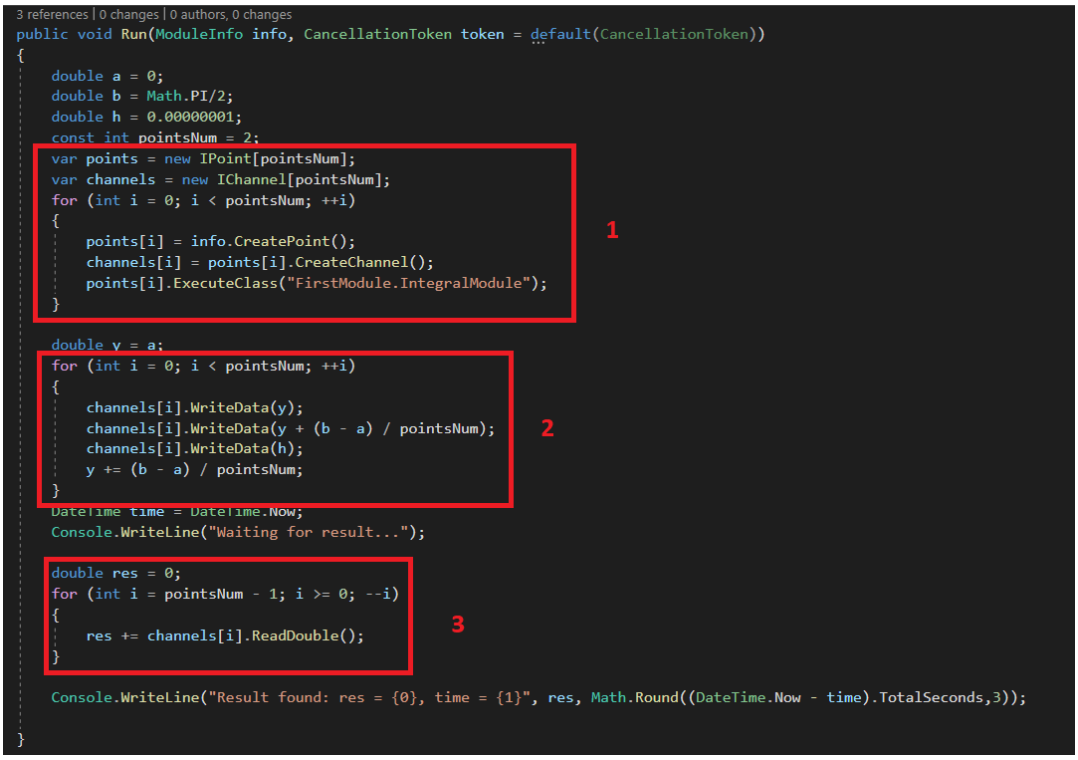


Рисунок 1 - Приклад АМ в ПАРКС.NET

Система PARCS-NET повністю відповідає засадам оригінальної концепції. Серед переваг реалізації можна виділити:

- Першопроходець в світі .NET [1];
- Незалежна від основного рішення розробка АМ через перевикористання NuGet пакетів [1212, 1313];
- Надшвидка передача даних через низькорівневість обраного протоколу мережевої комунікації;
- Контейнеризація компонент [66];
- Інтеграція з трьома найпопулярнішими хмарними провайдерами [6, 14, 15]
- Вичерпна документація.

Однак, серед недоліків констатуємо:

1. В основі усіх складових рішення лежить застарілий, залежний від ОС Windows фреймворк .NET Framework 4.6.1;

- Підтримка цього фреймворку завершилась в 2022 році [16], тобто платформа більше не отримуватиме важливих оновлень в сферах безпеки,

швидкодії, інтеграції і т. п. Окрім того, стосовно проблем з непідтримуваними версіями .NET не можна звертатись в підтримку Microsoft, що стає в нагоді в деяких виняткових ситуаціях.

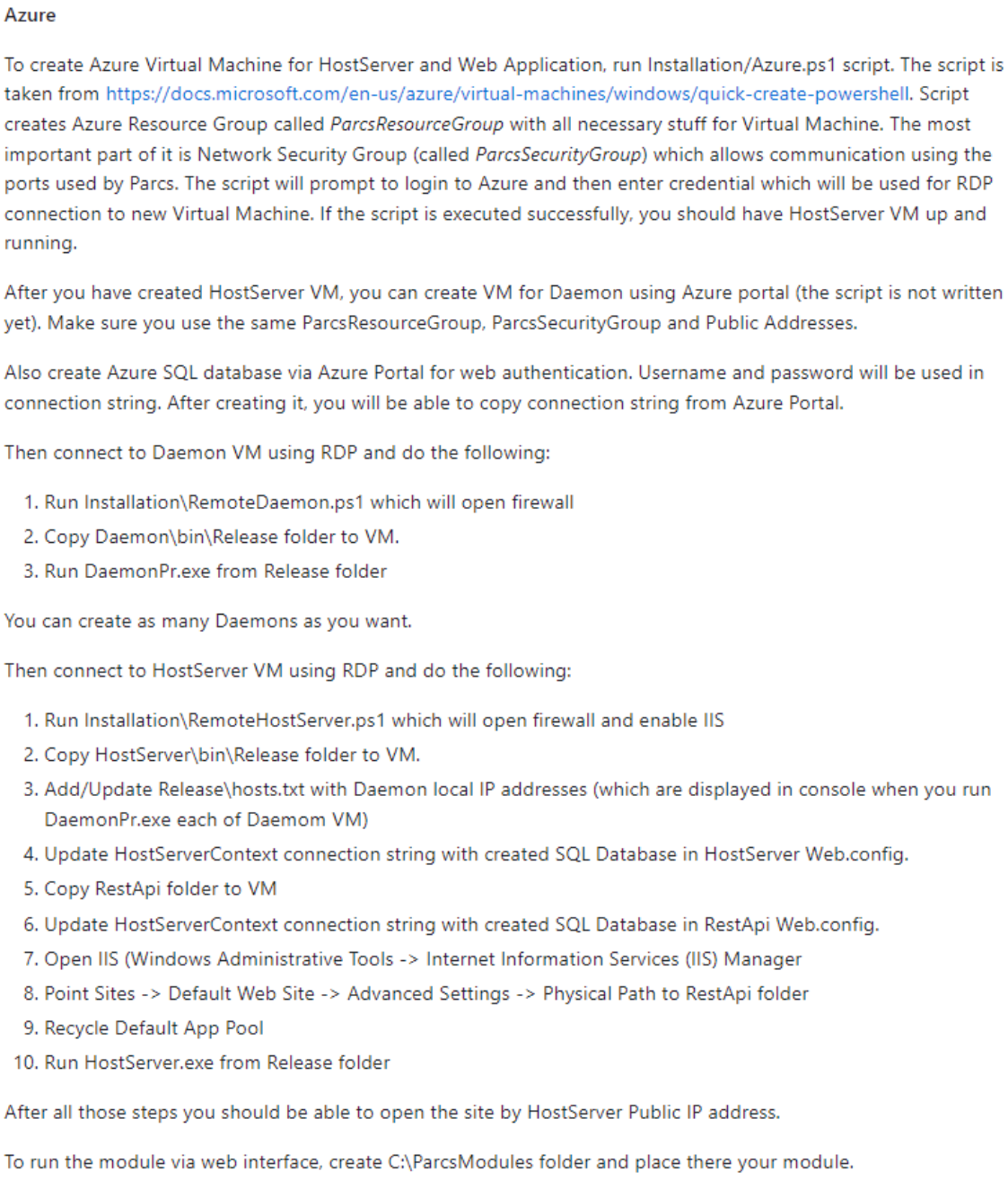
- Зав’язка під конкретну операційну систему накладає обмеження цієї системи на реалізацію ПАРКС. Окрім того, звужується вибір цільової інфраструктури для публікації («*Всі образи мають тег windowsservercore-1709, що означає, що вони створені на основі базового образу Windows Server Core 1709. Це дозволяє запускати їх на Google Cloud Platform.*» [6]).

- Новіші фреймворки відрізняються не лише назвою, а й внутрішніми складовими. Однією з таких є імплементація «гарячих» секцій коду (з англ. «hot paths»), прикладом якої є широко застосовна в ПАРКС рефлексія часу виконання програми. В порівнянні з .NET Framework 4.8, який вийшов трохи пізніше за 4.6.1, остання версія платформи досягає чотирикратного пришвидшення [17].

2. Цільова інфраструктурна модель – On-Premises або IaaS;

- В залежності від повноти відповідальності, вирізняють чотири моделі публікації застосунків – On-Premises, IaaS, PaaS та SaaS. Починаючи з On-Premises, в якій усе операційне навантаження лежить на плечах власника застосунку, і в порядку зростання до SaaS, повноваження поступово переходять до хмарного провайдера: в IaaS він забирає на себе управління фізичним центром обробки даних, мережу та ЕОМ; в PaaS – операційну систему та частково мережеві налаштування, хостинг і управління доступами; в SaaS часткові міграції PaaS стають повноцінними [18]. Чим вищим є рівень сервісу, тим менше менеджменту залучається в підтримку та налаштування застосунку зі сторони власника.

- З огляду на специфіку реалізації, ПАРКС-NET потребує повного контролю над операційною системою для успішної публікації. Це, втім, ускладнює його розгортання та підвищує поріг знайомства з технологією, чому доказ - відповідні громіздкі інструкції (див. рисунок 2).



Azure

To create Azure Virtual Machine for HostServer and Web Application, run Installation/Azure.ps1 script. The script is taken from https://docs.microsoft.com/en-us/azure/virtual-machines/windows/quick-create-powershell. Script creates Azure Resource Group called *ParcsResourceGroup* with all necessary stuff for Virtual Machine. The most important part of it is Network Security Group (called *ParcsSecurityGroup*) which allows communication using the ports used by Parcs. The script will prompt to login to Azure and then enter credential which will be used for RDP connection to new Virtual Machine. If the script is executed successfully, you should have HostServer VM up and running.

After you have created HostServer VM, you can create VM for Daemon using Azure portal (the script is not written yet). Make sure you use the same ParcsResourceGroup, ParcsSecurityGroup and Public Addresses.

Also create Azure SQL database via Azure Portal for web authentication. Username and password will be used in connection string. After creating it, you will be able to copy connection string from Azure Portal.

Then connect to Daemon VM using RDP and do the following:

1. Run Installation\RemoteDaemon.ps1 which will open firewall
2. Copy Daemon\bin\Release folder to VM.
3. Run DaemonPr.exe from Release folder

You can create as many Daemons as you want.

Then connect to HostServer VM using RDP and do the following:

1. Run Installation\RemoteHostServer.ps1 which will open firewall and enable IIS
2. Copy HostServer\bin\Release folder to VM.
3. Add/Update Release\hosts.txt with Daemon local IP addresses (which are displayed in console when you run DaemonPr.exe each of Daemom VM)
4. Update HostServerContext connection string with created SQL Database in HostServer Web.config.
5. Copy RestApi folder to VM
6. Update HostServerContext connection string with created SQL Database in RestApi Web.config.
7. Open IIS (Windows Administrative Tools -> Internet Information Services (IIS) Manager
8. Point Sites -> Default Web Site -> Advanced Settings -> Physical Path to RestApi folder
9. Recycle Default App Pool
10. Run HostServer.exe from Release folder

After all those steps you should be able to open the site by HostServer Public IP address.

To run the module via web interface, create C:\ParcsModules folder and place there your module.

Рисунок 2 – Інструкції з розгортання оригінальної версії PARCS-NET [19]

- Звичайно, запропоновані PowerShell скрипти можна, наприклад, загорнути в нові та стиснути перелік до одного-двох пунктів. І справді, з введенням поняття контейнерів в [6] і залученням підтримки сервісу створення віртуальних машин на основі шаблонів, кількість кроків зменшилась (див. рисунок 3). Однак, як безпосередній ресурс, віртуальні машини залишились, хоч потреби в такому ступені контролю і відповідальності більше нема.

Використання системи ПАРКС.NET на Google Compute Cloud

Для використання системи ПАРКС.NET на Google Compute Cloud необхідно виконати наступні кроки:

1. Створити віртуальну машину на основі образу: *windows-1709-core-for-containers*.
2. Запустити *Docker* контейнер для HostServer на основі образу: *../parcshostserver:windowsservercore-1709*. Також це можна зробити за допомогою скрипту запуску.
3. Налаштувати правила мережевого екрану.
4. Створити шаблон віртуальної машини для *Daemon*. У розділі Custom metadata прописати Powershell скрипт для запуску контейнера *Daemon* з параметром зі змінною оточення: EXTERNAL_LOCAL_IP_ADDRESS. Значення змінної, зовнішню IP-адресу можна отримати за допомогою HTTP запиту до Google API. Також потрібно вказати змінну оточення: PARCS_HOST_SERVER_IP_ADDRESS з IP-адресою машини, на якій запущено *HostServer*. На рис. 3 наведено приклад такого скрипта.
5. За допомогою групи віртуальних машин (Instance groups) створити і запустити необхідну кількість віртуальних машин на основі описаного вище шаблону. На рис. 1 маємо 4 віртуальні машини в групі.
6. Дочекатись, поки всі екземпляри *Daemon* підключяться до *HostServer*. Після цього в консолі *HostServer* з'являться повідомлення про підключення кожного екземпляру *Daemon* рис. 2.

Рисунок 3 – Інструкції з розгортання контейнер-орієнтовного PARCS-NET

3. Відсутність графічного інтерфейсу;

- Наявність зручного графічного інтерфейсу для роботи з системою – запорука кращого користувацького досвіду, про що свідчить PARCS-PYTHON [20], де можливість створювати модулі і задачі надається із браузера (див. рисунок 4). Варто також враховувати, що Python – інтерпретована мова. Через це, створювати АМ для неї можна інтуїтивно, з сирцевих кодів, в той час як в C# спершу необхідна компіляція в IL [21].

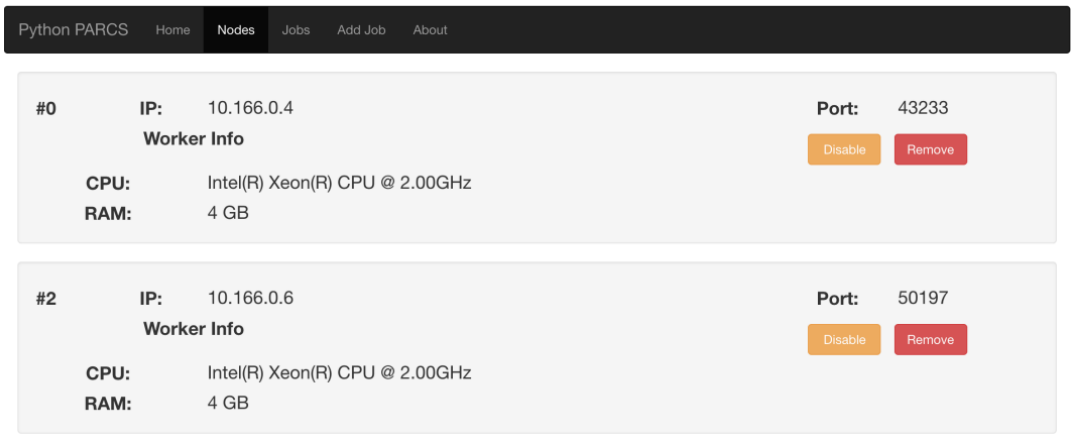


Рисунок 4 – Графічний інтерфейс системи PARCS-PYTHON

4. Неможливість програмного створення модулів;

- Серед відкритих операцій веб-інтерфейсу RestApi, який надає доступ до команд управління системою, для модулів виділено дві команди: «отримати перелік» і «виконати». Остання припускає, що модуль вже завантажено до файлової системи, на якій розгорнуто PARCS-NET, про що сказано в інструкції: «*To run the module via web interface, create C:\ParcsModules folder and place there your module*» (див. рисунок 2). З огляду на такий підхід, система не може вважатись самодостатньою. Натомість вона є функціонально залежною від середовища публікації, внаслідок чого знижується її здатність до портування.

5. Синхронний код.

- За умови послідовного виконання програмних інструкцій потік очікує на завершення кожної наступної такої замість того, щоб в цей час займатись іншою корисною діяльністю. Це виснажує системні ресурси і в перспективі призводить до сповільнення роботи програми. Використання синхронного коду за наявності асинхронних альтернатив (напр. async/await, починаючи з .NET Framework 4.5) негативно впливає на можливість масштабування програми, до чого особливо чутливі багатопотокові застосунки [22].

Загалом, PARCS-NET – потужна платформа для виконання паралельних алгоритмів на мережі, нечисленні недоліки якої в більшості своїй зумовлені обмеженістю наявності відповідних технологічних засобів на час її написання.

1.6. PARCS-WCF

PARCS-WCF – наразі найновіша версія ПАРКС для .NET [2]. Насамперед система будувалась як адаптація вже існуючої для моделювання операцій реляційної алгебри вибору, однак через нову платформу в основі рішення, зміни до абстракцій та удосконалення деяких процесів, PARCS-WCF набула самостійності і тому має розглядатись окремо. Запропонована система складається з таких компонент:

- WCF веб-сервісу DaemonHost, який під час запуску створює WCF-прив’язки типу TCP та NetTcpBinding, де перша використовується для мережевої комунікації, друга – локально для IPC. DaemonHost публікує три сервіси: IDaemonService, що задає операції логічного рівня (напр. створення точки, знищення КП) і допоміжні (напр. надсилання файлу); IPointService, що задає комунікацію між точками (напр. створення каналу, відправка даних); та Metadata, що надає діагностичну інформацію про сам WCF сервіс. Розкриття методів сервісів відбувається за допомогою специфічних для WCF атрибутів (ServiceContract та OperationContract). Разом з адресами та типами прив’язок вони визначають кінцеві точки, що автоматично співставляються засобами WCF. За замовчуванням, дані між кінцевими точками передаються у форматі JSON.
- консольних застосунків, що описують АМ. Через модульні абстракції кожен такий застосунок має змогу підняти екземпляр веб-сервісу та підключитись до існуючих, задати з їх допомогою логічну структуру КП та виконати необхідні обчислення.

На рисунку 5 наведено базисний модуль обчислення інтегралу, приклад АМ в PARCS-WCF. Як бачимо, АМ тут не закріплені за контрактом інтерфейсу і можуть бути створені в довільному місці, з якого доступні класи ControlSpace, Point і PointStartInfo, а також переліки PointType і ChannelType. На даний момент всі зазначені сутності розміщені в бібліотеці класів Parcs, на яку як на проект можна посилатись лише всередині рішення ParcsSharp. Потенційно цю бібліотеку можна було б опублікувати як NuGet пакет.

Прямокутником з номером «1» виділено блок коду створення КП (топології мережі), прямокутником з номером «2» - розподіл вхідних даних між точками («наповнення» логічної структури КП), прямокутником з номером «3» - збір вихідних даних з точок (агрегація результатів обчислень). Хоч візуально код модуля є доволі схожим на аналогічний в PARCS-NET (див. рисунок 1), все ж є суттєві відмінності: явне створення об’єкту КП, можливість задавати тип каналів і точок (локальні або мережеві) та прихована робота з каналами (створюються всередині методу CreatePointAsync).

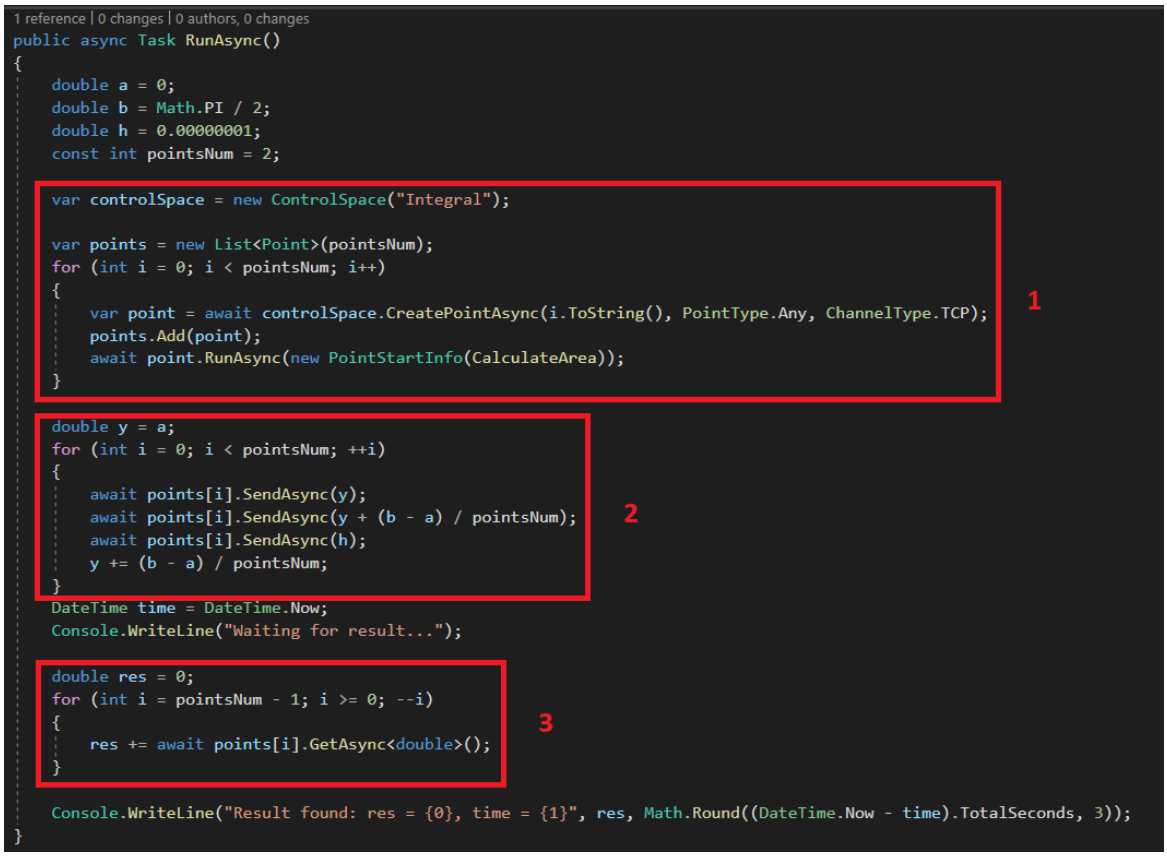


Рисунок 5 – Приклад АМ в PARCS-WCF

Система PARCS-WCF повністю відповідає засадам оригінальної концепції. Серед переваг реалізації можна виділити:

- Асинхронність;
- Можливість переключатись між протоколами передачі даних в залежності від фізичного розташування точки;
- Можливість встановлення захищеного каналу шляхом провадження аргументу Security до налаштувань прив’язок;
- Можливість перевантажувати механізм серіалізації даних шляхом реалізації інтерфейсу IParcsSerializer;
- Можливість реалізувати свій планувальник завдань на основі діагностичної інформації, що надає сервіс метаданих;

Однак, серед недоліків констатуємо:

1. Windows-залежний .NET Framework 4.5, підтримка якого завершилась в 2016 році (див. недоліки PARCS-NET п.1);
2. Відсутність графічного інтерфейсу (див. недоліки PARCS-NET п.3);

3. Відсутність моделі публікації або відповідних рекомендацій.

- Сервіси WCF можуть бути опубліковані в одному з трьох варіантів: як консольний застосунок, Windows Service або в середовищі IIS [23]. Наразі DaemonHost в PARCS-WCF – це консольний застосунок, тож розгорнути його можна або локально, або на одній чи декількох віртуальних машинах. Як альтернативна, якщо налаштувати DaemonHost під середовище IIS, відкривається опція Azure App Service, що на відміну від інших варіантів є PaaS рішенням. У будь-якому випадку процес повністю мануальний, а тому складно масштабований і трудомісткий.

У кінцевому підсумку PARCS-WCF доводить, що концепції [11] можна реалізовувати по-різному. Внутрішня складова системи має суттєві відмінності, як порівняти з PARCS-NET, при цьому ефективністю перша другій не поступається. Крім того, PARCS-WCF пропонує низку вдосконалень в плані швидкості та гнучкості, хоч і не вирішує всіх проблем оригінальної імплементації.

РОЗДІЛ 2. РОЗРОБКА PARCS-NET-K8

Запропонована в роботі реалізація технології програмування ПАРКС, що в ході розробки отримала назву PARCS-NET-K8, мала на меті дати відповіді на відкриті питання, які залишили по собі попередні версії системи. Детальний аналіз пов'язаних праць та вихідного коду дозволив локалізувати пріоритетні напрямки розвитку платформи та зосередити зусилля на її нині актуальних проблемах. PARCS-NET-K8 робить акцент на сучасних технологіях та стандартах індустрії, при цьому дотримуючись історичних засад та повторно використовуючи попередній досвід в сфері.

2.1. Проектування архітектури

Насамперед маємо означити два традиційні процеси. Перший, умовно демон, відпрацьовує основні функції КП: створює точки і канали, приписує АМ точкам, підтримує взаємодію між ними. Демон виконує запуск алгоритмічних модулів обчислювальних задач, а також тестування продуктивності машин. Його ресурсні потреби явно вимагають окремої стратегії масштабування, звідки маємо виокремлення в окрему компоненту. Другий, умовно хост, – централізована служба для управління та обліку задач і точок. Хост зберігає інформацію про підключені вузли, продуктивність та кількість точок на кожному вузлові, і згідно цієї інформації виділяє машини для створення нових точок. Також хост відмічає видалення точок, початок та завершення задач [9].

Однією з цілей цієї версії ПАРКС було також надати графічний інтерфейс, котрий, як було зазначено в аналізі попередніх робіт, донині був відсутнім. Ця складова рішення, умовно портал, могла б бути вбудована в хост або ж реалізована як окремий проект. Розглянемо переваги і недоліки обох підходів.

Серед переваг першого можна виділити наступне:

- В рішенні менше рухомих частин, через що, наприклад, під час відлагодження не потрібно перемикатись між різними проектами. Це додає інтуїтивності в процес роботи з кодом;

- Дві складові спілкуються в межах одного процесу, що знижує затримку між ними і покращує загальний користувацький досвід;
- Наявність одного фізичного суб’єкта тестування спрощує написання функціональних тестів.

Водночас, недоліками тут є:

- Щільне зв’язування (з англ. «tight coupling»), через що модифікувати одну частину рішення і не задіти при цьому іншу стає вкрай складно або навіть неможливо;
- Обмежена масштабованість (будь-які зміни обчислювальних можливостей тут позначаються на обидвох складових);
- Нагромадження коду і ускладнення навігації ним з часом.

На противагу, переваги розподіленої версії такі:

- Слідування одному із фундаментальних принципів програмування – розмежуванню повноважень (з англ. «separation of concerns»). Якщо всі компоненти мають свої чітко визначені обов’язки, кожна може сфокусуватись на власних проблемах, а не на зовнішніх залежностях;
- Можливість застосовувати до двох частин різні, не обов’язково сумісні, але оптимальні для своїх сценаріїв мови програмування, фреймворки тощо;
- Зміни до одного проекту, навіть критичні, не мають жодного впливу на інший;
- Незалежна масштабованість.

Зазначені переваги дістаються ціною таких недоліків:

- Пропорційно кількості проектів зростає і складність системи, а з нею – час відлагодження, пошуку помилок, а також поріг цілісного сприйняття;
- Додається мережева комунікація, що веде до затримок і негативно впливає на швидкодію;
- Збільшується інфраструктурний відбиток, а з ним – коштовність рішення і складність операційного процесу [24].

Аналізуючи перелік вище, складно дійти однозначного висновку. З одного боку, для ПАРКС істотно важливими є швидкодія та простота тестування, з іншого - незалежність розгортання, розширюваність та гнучкість. З огляду на те, що об’єктом вибору є місце графічного представлення в системі, а не функціональна складова, накладні витрати тієї чи іншої моделі тут некритичні. У кінцевому підсумку було віддано перевагу розподіленій структурі.

В контексті вищенаведеного порівняння варто також відзначити правильність рішення розмежувати хост і демонів. Оскільки останні на собі розміщують точки КП, а тому мають відмінні від хоста потреби в плані обчислювальної потужності, рішення було цілком обґрунтованим, адже розподілена модель, серед іншого, пропонує незалежність масштабування.

Отже, маємо хост, демони і портал – основні компоненти PARCS-NET-K8 (див. рисунок 6), в чому можна прослідити наявну схожість з PARCS-NET, де архітектура була аналогічною. За тими ж принципами було також створено і бібліотеку класів з абстракціями, необхідними для незалежної розробки алгоритмічних модулів. Додатково, для хоста було виділено базу даних, а також спільне з демоном сховище, де вони могли б зберігати бінарні коди модулів й файли з вхідними та вихідними даними задач,. Це в підсумку дозволило позбавитись залежності від файлової системи середовища виконання, де донині такі ресурси розміщувались.

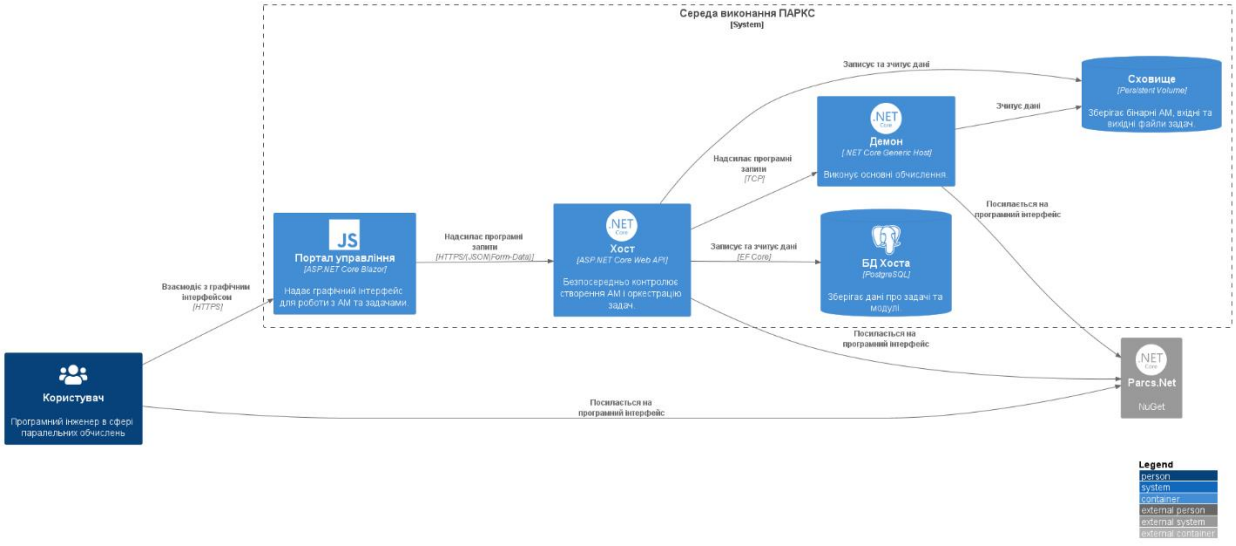


Рисунок 6 – Архітектура PARCS-NET-K8 (C4, Рівень «Контейнер» [2525])

2.2. Вибір фреймворку

Спершу робочим фреймворком для демона, хоста, спільної бібліотеки класів і порталу було обрано .NET 6, нині останню збірку платформи з приміткою LTS (з англ. «Long Term Support»). Вихід її стабільної версії датовано листопадом 2021 року, а підтримка закінчується наприкінці 2024, близько імовірного виходу .NET 8, наступної LTS. Водночас життєвий цикл найсучаснішої збірки, .NET 7 (2022), простягається лиш до травня 2024 [26].

Незважаючи на це, було проведено детальний аналіз нововведень в .NET 7, зокрема удосконалень в плані швидкодії [27], де з-поміж іншого згадується рефлексія. Остання широко застосовується в реалізаціях ПАРКС-технології програмування з незалежною розробкою АМ, яких є більшість. PARCS-NET-K8, як і PARCS-NET, пропонує побудову модулів через спільні абстракції, тож цей фактор пришвидшення (напр. Activator.CreateInstance – 2.276 мікросекунд проти 3.827 в .NET 6, AssemblyName.FullName – 2.010 мікросекунд проти 3.423 в .NET 6) став визначальним. Раннє ж завершення підтримки сьомої версії платформи не є критичною проблемою, адже процес оновлення проекту до наступної версії став практично елементарним після злиття історично різних розгалужень .NET (Framework, Standard, Mono) в єдиний .NET 5 в 2019 (див. приклад в [28]).

Як порівняти ж використаний в роботі сучасний .NET 7 (2022) зі згаданими раніше .NET Framework 4.5 (2012) та .NET Framework 4.6.1 (2015), маємо суттєві відмінності:

- Багатоплатформність: .NET 7 дозволяє створювати та публікувати додатки на macOS, Linux, iOS та Android. Таким чином, зникає залежність від операційної системи Windows;
- Швидкодія: реалізація .NET 7 набагато швидша за .NET 6, що вже казати про .NET Framework. Можна виокремити миттєвий запуск застосунків, вдосконалений механізм збору сміття (з англ. «garbage collection»), численні апаратні оптимізації тощо;
- Збагачені мовні засоби: C# 10 пропонує масу мовних функцій для збільшення ефективності написання коду. Серед таких засобів можна виділити

співставлення шаблонів, нову структуру record, NULL-перевірки часу компіляції тощо;

- Вдосконалений користувацький досвід розробки: .NET 7 надає просунуті утиліти для відлагодження та аналізу коду, автоматичного доповнення, профілювання та інтеграції з іншими платформами;
- Краща безпека: в .NET 7, як і в кожній наступній версії фреймворку, більше захисту від розповсюджених вразливостей;
- Модульність: .NET Core має набагато легшу основу проти .NET Framework. Це досягається завдяки тому, що Core побудований навколо модулів – NuGet пакетів. Це робить його більш привабливим, наприклад, з точки зору контейнеризації;
- Відкритий доступ: починаючи з перших версій .NET Core (2016), увесь код фреймворку став відкритим. За потреби з його деталями можна ознайомитись у єдиному публічному репозиторії [2929, 30].

2.3. Деталі реалізації хоста

Хост побудовано на базі ASP.NET Core, де-факто стандарті в розробці веб-сервісів на платформі .NET. З його допомогою можна створювати контракт-орієнтовні сервіси (RPC), REST, застосунки в режимі реального часу, MVC, клієнтські додатки тощо. Все це завдяки численним інтеграціям з сучасними технологіями та хмарою, вбудованою ін’єкцією залежностей (з англ. «dependency injection»), логуванням, модульному обробнику HTTP запитів, підтримці хостингу на Kestrel, Nginx, Apache, Docker та IIS. А його гнучкість та легкість у використанні можна продемонструвати на Minimal API, що були представлені з виходом .NET 6: створити повноцінний веб-сервер для обробки HTTP запитів можна всього-на-всього в три рядки [31].

Хост являє собою RESTful Web API [32], що базується на REST (з англ «Representational State Transfer») - архітектурному стилі та підході до побудови комунікацій, який користується популярністю у веб-розробці через ефективне використання мережі (кожен запит надає всю необхідну інформацію для його обробки; кожна відповідь надає лиш інформацію про запитуваний ресурс).

Слідуючи принципам зазначеного стилю, хост має уніфікований інтерфейс (модулі та задачі доступні за унікальними гіперпосиланнями через використання методів протоколу HTTP – див. рисунок 7), клієнт-серверну архітектуру (за графічний інтерфейс та побудову запитів відповідає портал; за доступ до даних і маніпуляцію над ними – сервер), не зберігає стан (кожен наступний запит не є залежним від попереднього), припускає кешування (наприклад, статусів задач) та багатошарову архітектуру (запити на створення задач можуть пропагуватись далі, на демонів).

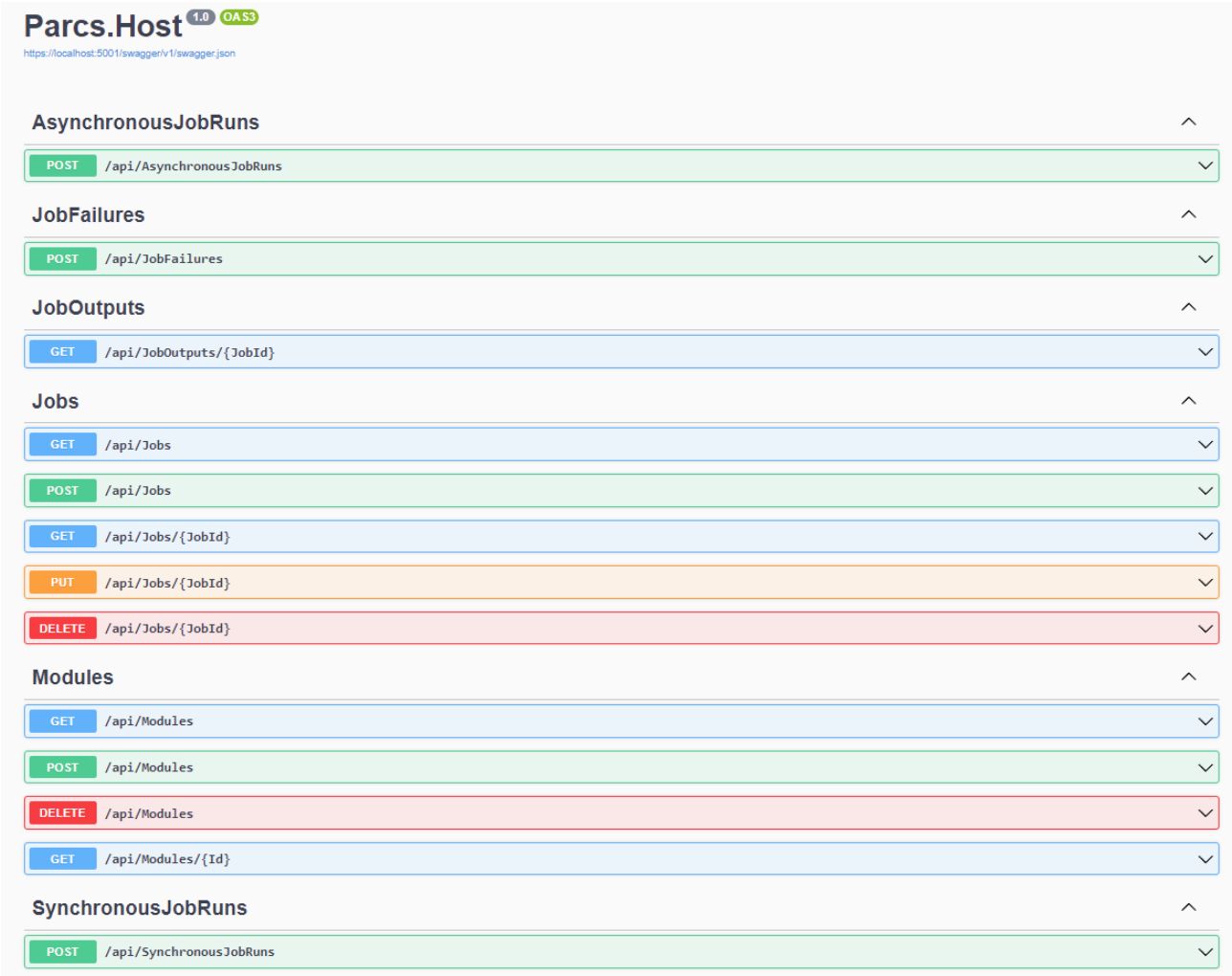


Рисунок 7 – Веб-інтерфейс хоста

Як видно з рисунку 7, всього розкривається чотири ресурси: модуль, задача, її синхронний та асинхронний запуск. Розберемо окремі операції:

- *POST api/Modules* – створення модуля. Приймає на вхід колекцію бінарних файлів. Правила валідації цього запиту вимагають непустої колекції, причому розширення хоча б одного з наданих файлів має бути DLL (з англ. «Dynamic Link Library» - формат, що використовується для збереження та поширення скомпільованого коду). У разі успішної обробки файли зберігаються в директорії спільного сховища, що закріплена за модулем, а у відповідь повертається унікальний ідентифікатор, за яким на цей модуль можна посилатись;
- *POST api/Jobs* – створення задачі. Приймає на вхід ідентифікатор модуля, вхідні файли, назву збірки та класу для точки входу. Правила валідації цього запиту доволі комплексні. Спершу робиться перевірка на наявність ідентифікатора, який має посилатись на зареєстрований модуль (модуль вважається зареєстрованим, якщо в сховищі існує відповідна непуста директорія з бінарними файлами). Після цього, якщо модуль знайдено, за вказаною назвою збірки виконується пошук серед файлів в директорії модуля. Якщо збірку також знайдено, перевіряється присутність в ній зазначеного класу (див. рисунок 8).
- *POST api/JobFailures* – створення помилки задачі. Ця операція використовується виключно демонами: у разі виникнення помилок в момент виконання алгоритмічного модуля, відповідний демон має сповістити хоста. Під час обробки цього сповіщення створюється новий статус задачі, Failed, а усім доступним демонам надсилається команда перевірити чи виконується на них ця задача і, якщо так, то відмінити її;
- *GET api/Jobs/{JobId}* – перегляд задачі. Приймає на вхід ідентифікатор. Якщо задачу знайдено, повертає наявні дані про неї, що включають історію статусів, батьківський модуль та помилки;
- *DELETE api/Jobs/{JobId}* – видалення задачі. Приймає на вхід ідентифікатор. Якщо задачу знайдено і вона вже в процесі обробки, відміняє виконання пов’язаних операцій на всіх рівнях, після чого видаляє відомості про неї з бази даних. Якщо задача в черзі на виконання, видаляє її з черги;

- *POST api/SynchronousJobRuns* – синхронний запуск задачі. Приймає на вхід ідентифікатор задачі, кількість точок та словник аргументів. Запитувана кількість точок вона має бути позитивним числом, а задача - існуючою. Увесь час виконання ініціатор виклику очікує на завершення, звідки й назва. По завершені у відповідь надсилається фінальний статус задачі;
- *POST api/AsynchronousJobRuns* – асинхронний запуск задачі. Відрізняється від попередньої операції тим, що безпосереднє виконання тут відбувається в фоновому режимі: клієнт отримує ідентифікатор створеної задачі, інформацію про яку він може періодично запитувати. По завершені роботи за вказаним гіперпосиланням надсилається оповіщення з ідентифікатором. Звідси єдиний відмінний від попереднього набору параметр, CallbackUrl.

Рисунок 8 – Приклад неуспішного запиту на POST api/Jobs

Варто відзначити механізм обробки запитів. Усі вони на рівні контролерів загортаються в моделі відображення та зміни стану відповідно до концепції CQS (з англ. «Command Query Separation»), після чого передаються на вхід посереднику (з англ. «Mediator»). Як і в однойменному поведінковому шаблоні проектування [33], посередник координує взаємодію об’єктів: вирішує як і коли вони спілкуються один з одним на основі вхідних даних та способу їх передачі. Завдяки цьому підходу вирішується проблема вибуху залежностей, класи стають

менш зв’язними та більш перевикористовуваними, тестування - простішим, однак навігація кодом - менш очевидною. Рішення вдатись до цієї технології, що в світі .NET в першу чергу представлена бібліотекою MediatR, було прийнято через зростаючу складність компонент під час розробки.

Як і в більшості реалізацій ПАРКС, в проекті широко застосовується рефлексія, зокрема для роботи з алгоритмічними модулями. Вперше вона зустрічається в прошарку валідації запитів на запуск задачі: серед бінарних файлів заданого алгоритмічного модуля виконується пошук вказаної збірки, після чого, як збірку знайдено, середовище виконання завантажує її в пам’ять процесу, де за допомогою вбудованого System.Reflection.MetadataLoadContext інспектує її вміст і перевіряє наявність зазначеного класу. MetadataLoadContext слугує виключно для аналізу метаданих, але не запуску коду.

Протилежна ситуація виникає під час безпосереднього виконання алгоритмічних модулів. На відміну від .NET Framework, де ізоляція бібліотек реалізована через програмні домени, в .NET Core вводиться поняття контексту завантаження. За замовчуванням, застосунки .NET Core мають один такий контекст - AssemblyLoadContext.Default, в який в момент запуску програми поміщуються усі її залежності. Однак, якщо виникає потреба розділити цей простір завантажених бібліотек з іншими динамічно визначеними збірками, відразу постає питання стабільності додатка, адже нерідко такі збірки надходять з ненадійних джерел, мають різні версії або конфліктуючі імена. В таких випадках реалізовують спеціальні контексти, власне один з яких і було створено в описуваному проекті. Це дозволило ізолювати динамічні алгоритмічні модулі та мінімізувати їх вплив на роботу системи.

Окремої уваги також заслуговує порядок обробки запитів на асинхронний запуск задач. Використання стандартного Task.Run тут не є бажаним через природу веб-застосунків ASP.NET Core, адже кожного разу для виконання відповідного делегату виділяється потік зі спільного пулу потоків. Такі виділення поступово виснажують системні ресурси та порушують усталені евристики, оскільки з деякого моменту фіксована кількість початково запущених

потоків закінчується і кожен наступний має бути вже фізично заново створений. Це в перспективі призводить до потокового голоду (з англ. «thread starvation»), за якого система не встигає або не в змозі запускати нові потоки, через що подальші запити оброблюються в міру можливостей, із затримкою. В екстремальних випадках за таких умов система може взагалі перестати реагувати на будь-які звернення.

Саме тому фонове виконання задач на хості відбувається за іншою моделлю - виробника і споживача (з англ. «publisher/subscriber»): після створення сутності задачі команда на її запуск додається до внутрішньої потокобезпечної черги, що реалізовано засобами вбудованої абстракції для міжпроцесної взаємодії System.Threading.Channel [34]. Обробка доданої команди відбувається в окремому фоновому сервісі, який послідовно зчитує команди з черги і конвертує їх в запити синхронного запуску задач. Ініціатор асинхронного виклику при цьому не очікує на їх завершення, а натомість миттєво отримує ідентифікатор для відстеження прогресу і закриває підключення. Після завершення виконання запланованої задачі результати надсилаються на вказане в тілі запиту посилання.

2.4. Деталі реалізації демона

Демон – консольний застосунок з визначеним у ньому загальним .NET хостом, котрий в термінології .NET є об’єктом, що енкапсулює ресурси додатку та контроль над його життєвим циклом. Цей легковаговий механізм надає консольним застосункам ін’єкцію залежностей, логування, конфігурацію, контроль завершення роботи та фонові сервіси. Такими сервісами для демона є два сервери: внутрішній (ITC) та мережевий (TCP). Обидва призначені для обробки операцій на точках в системі ПАРКС.

Після запуску сервери починають активно очікувати на нові підключення: TCP - за наперед заданим портом, ITC – через міжпотоковий канал. Як тільки з’являється клієнт, його мережевий потік загортається в канал КП і передається на обробку в окремий сервіс. Як і в PARCS-NET, клієнт зобов’язаний на початку

серії повідомлень надіслати сигнал, що визначає спосіб обробки. Серед таких сигналів: ініціалізація задачі (зчитування аргументів й ідентифікаторів задачі та модуля, налаштування можливості відміни задачі), виконання класу (завантаження зі сховища в пам’ять та динамічний запуск бінарних файлів модуля), відміна задачі (виклик методу Cancel на токені, що закріплено за задачею), зміна протоколу (переключення на локальний канал), закриття каналу.

2.5. Деталі реалізації порталу

Портал, як і хост, є веб-додатком на основі ASP.NET Core. При цьому, на відміну від першого, він є більш орієнтованим на безпосередню взаємодію з користувачем, а не комунікацію між сервісами. Для забезпечення такої взаємодії був використаний компонентно-орієнтований UI-фреймворк Blazor. Основна ідея подібних технологій (до яких також відносяться React, AngularJS, VueJS) полягає в повторному використанні окремих фрагментів візуальної складової застосунків (панелей, форм, таблиць, графіків тощо). Характерною ознакою саме цієї бібліотеки є можливість побудови елементів інтерфейсу без залучення мови програмування JavaScript, замість якої тут виступає C#, що привносить потужність платформи .NET у клієнтський код.

Фреймворк має дві моделі хостингу: Blazor WebAssembly та Blazor Server. В першому випадку додаток повністю завантажується в пам’ять процесу веб-браузера та виконується на стороні клієнта, після чого запити до сервера надходять лише у разі необхідності взаємодії зі сторонніми сервісами. Такий підхід застосовується коли серед вимог до веб-додатку є забезпечення високої швидкості взаємодії із застосунком та слабка залежність від стану підключення до мережі. Недоліком є вивантаження серверного коду на кінцевий користувацький пристрій, що теоретично може призвести до проблем з безпекою за рахунок декомпіляції. Недоліки Blazor WebAssembly визначають переваги Blazor Server. Останній інтегрується зі стандартним ASP.NET Core, що дозволяє повторно використовувати бізнес-логіку сторони сервера в графічних компонентах Razor.

Інтерактивний режим та двосторонній обмін даними в Blazor забезпечується встановленням стійкого з'єднання на основі технології SignalR в момент створення користувацької сесії. SignalR – бібліотека для ASP.NET Core, що надає зручний програмний інтерфейс для клієнт-серверної взаємодії у режимі реального часу: в залежності від можливостей тієї чи іншої сторони технологія обирає оптимальний тип двосторонньої взаємодії з пріоритетного списку транспортів. До цього списку належать Long Polling, що періодично встановлює тривале одностороннє підключення для передачі нових даних у разі їх появи, Server Sent Events, що підписується на отримання оновлень через зарезервовані відкриті канали, та WebSocket, що передбачає встановлення двонаправленого з'єднання, яким для обміну інформацією за потреби може користуватись як клієнт, так і сервер [35]. У якості моделі для розробки веб-порталу системи було віддано перевагу Blazor Server в першу чергу через кращу портативність та відсутність проблем з безпекою.

2.6. Огляд бази даних

Для зберігання інформації про зареєстровані модулі та виконувані або вже виконані задачі було спроектовано окрему базу даних. Через негетерогенну структуровану природу цих даних, а також наявність прямого відношення «один до багатьох» між задачами та модулями, базу було визначено як реляційну. В якості системи управління обрано PostgreSQL, що останнім часом набула популярності через свою високу швидкодію, розширюваність, надійність, кросплатформність та портативність. У якості ж прошарку між СУБД та застосунком задіяно засоби Entity Framework Core – традиційного для .NET об'єктно-реляційного відображення, що надає такі просунуті функції як проектування на основі коду, написання інтегрованих в мову запитів (LINQ), відстеження змін сутностей, вбудовані транзакції тощо.

В основу схеми бази даних вочевидь лягли дві основні сутності: модулі та задачі. Втім, до них згодом було також додано ще дві таблиці: помилки та статуси задач (див. рисунок 9). Таке рішення сприяло підтриманню нормальної

форми Бойса-Кодда, адже і помилок і статусів у задачі може бути багато, а тому зберігати їх разом з основною сутністю означає або порушувати першу нормальну форму, або перезаписувати останнє оновлення і втрачати історичну інформацію. Окрім того, впровадження дозволило убезпечити базу від модифікацій: записи в таблиці лише вставляються і ніколи не змінюються.

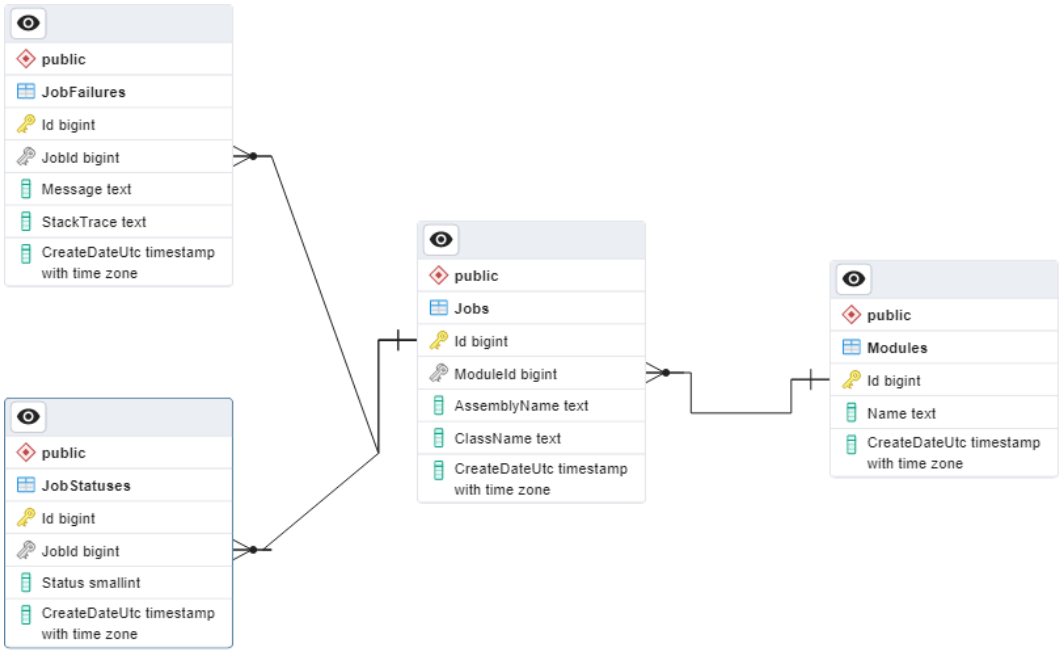


Рисунок 9 – Схема бази даних

2.7. Огляд сховища

Як було зазначено раніше, спільне для демона та хоста сховище абстрагує від них файлову систему, таким чином відв’язуючи ПАРКС від зовнішньої середи. На рисунку 10 зображено структуру цього сховища: в директорії Jobs зберігаються вхідні та вихідні дані задач, а в Modules – бінарні файли модулів. Знаючи лиш ідентифікатор модуля, у цьому фіксованому каталозі можна легко знайти всі необхідні для його виконання збірки. Знаючи ідентифікатор задачі – всі файли, що необхідні для та що створені внаслідок її роботи.

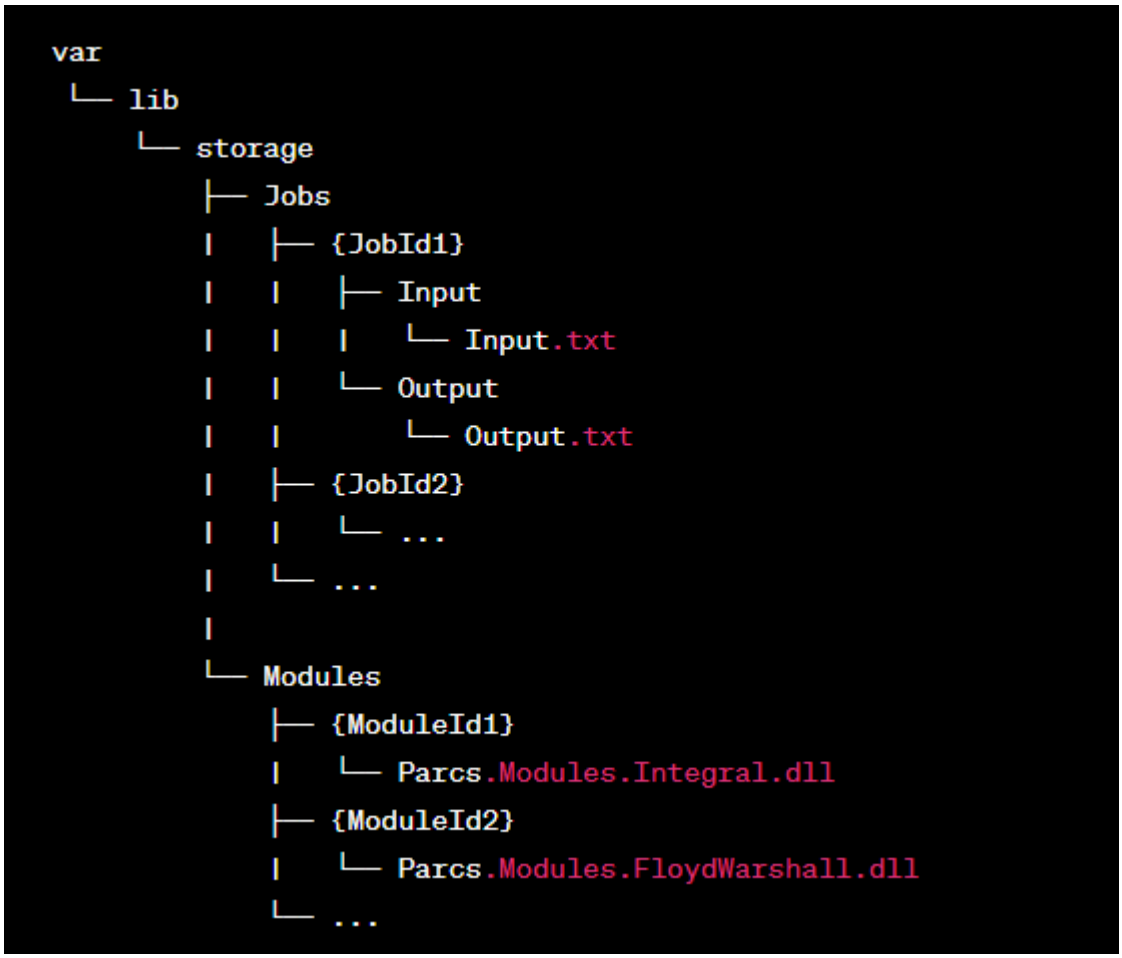


Рисунок 10 – Структура спільного сховища

2.8. Огляд бібліотеки Parcs.Net

Parcs.Net [3636] – бібліотека класів, що опублікована як NuGet пакет. Її виокремлення та поміщення в публічний простір має на меті єдину ціль: надати користувачеві доступ до ієрархічної структури віртуальних ПАРКС-машин [9], при цьому приховуючи деталі реалізації самої системи. За умови, що розробник АМ має змогу реєструвати АМ в розгорнутому рішенні, їй або йому не потрібно локально завантажувати весь програмний код та встановлювати проектні залежності. Натомість, достатньо буде створити стандартну бібліотеку класів та додати до неї посилання на Parcs.Net, згенерувати бінарні файли на основі написаних сирцевих кодів і нарешті зареєструвати новий модуль в системі.

Такий підхід в проектуванні називається «архітектура на основі плагінів». Відповідно до нього, виконувана програма не знає деталей реалізації динамічно під’єднаних модулів, але визначає єдиний інтерфейс, якому всі вони мають

слідувати. Завдяки цьому інтерфейсу вона стає дуже гнучкою та легко розширюваною, адже може завантажувати і виконувати конкретні реалізації, при цьому ніяк від них не залежачи.

В [3737] наведено приклад плагін-орієнтованого застосунку на базі .NET. Відзначимо, що реалізація тут значною мірою ґрунтується на вже раніше згадуваному класі AssemblyLoadContext, який і було використано в рішенні. Окрім того, відповідно до розділу «Reference a plugin interface from a NuGet package», у випадку розміщення плагін-інтерфейсу в NuGet пакеті необхідно в плагін-проекті до посилання на пакет додати параметр ExcludeAssets зі значенням «runtime». Це налаштування запобігає копіюванню бінарних файлів NuGet пакету в директорію зі збіркою плагін-проекту, і таким чином змушує виконувану програму використовувати її власну версію цього пакету. Відповідно, розробники плагін-проектів, тобто АМ в ПАРКС, мають враховувати це єдине незначне обмеження.

Щодо вмісту Parcs.Net, маємо тільки необхідні контракти: IArgumentsProvider для доступу до аргументів задачі зсередини модулів, IChannel як абстракція каналу, IPoint як абстракція точки, IModule як абстракція модуля та IModuleInfo як узагальнення знань модуля про середовище. Окрім того, IInputReader як сервіс доступу до вхідних даних задачі, IOutputWriter як сервіс збереження вихідних даних задачі, а також описаний в 2.4 перелік Signal.

Для порівняння з PARCS-NET та PARCS-WCF на рисунку 11 наведено базисний модуль обчислення інтегралу, приклад АМ в PARCS-NET-K8. Прямокутником з номером «1» виділено блок коду створення КП (топології мережі), прямокутником з номером «2» - розподіл вхідних даних між точками («наповнення» логічної структури КП), прямокутником з номером «3» - збір вихідних даних з точок (агрегація результатів обчислень), прямокутником з номером «4» - знищення КП.

Як можна бачити, візуально представлення є дуже подібним до аналогічного в PARCS-NET (див. рисунок 1): схожим є набір параметрів методу та процес створення каналів і точок. Прослідковуються й запозичення з PARCS-

WCF (див. рисунок 5): код є асинхронним, а запуск класів на точках - типізованим (хоч доступним є і перевантаження з повним ім'ям збірки).

Втім, маємо й відмінності. По-перше, збереження та зчитування файлів відбувається не напряму через System.IO, а через абстракції IInputReader та IOutputWriter. По-друге, оскільки АМ тепер виконується не в консольному застосунку, котрий можна було б запустити напряму, а всередині віддаленого веб-сервісу, аргументи зчитуються не з командного рядка, а через абстракцію IArgumentsProvider. До останнього також додається метод-розширення, що дозволяє відобразити словник параметрів у користувацький тип. По-третє, маємо можливість програмно видаляти точки. Щоправда, це не є обов'язковим, адже після закінчення роботи методу всі вони видаляються автоматично.

```
public async Task RunAsync(IModuleInfo moduleInfo, CancellationToken cancellationToken = default)
{
    var moduleOptions = moduleInfo.ArgumentsProvider.Bind<ModuleOptions>();

    double a = 0;
    double b = Math.PI / 2;
    double h = moduleOptions.Precision ?? 0.00000001;

    var pointsNumber = moduleInfo.ArgumentsProvider.GetPointsNumber();
    var points = new IPoint[pointsNumber];
    var channels = new IChannel[pointsNumber];

    for (int i = 0; i < pointsNumber; ++i)
    {
        points[i] = await moduleInfo.CreatePointAsync();
        channels[i] = await points[i].CreateChannelAsync();
        await points[i].ExecuteClassAsync<WorkerModule>();
    }

    double y = a;
    for (int i = 0; i < pointsNumber; ++i)
    {
        await channels[i].WriteDataAsync(y);
        await channels[i].WriteDataAsync(y + (b - a) / pointsNumber);
        await channels[i].WriteDataAsync(h);
        y += (b - a) / pointsNumber;
    }
    DateTime time = DateTime.Now;
    Console.WriteLine("Waiting for result...");

    double result = 0;
    for (int i = pointsNumber - 1; i >= 0; --i)
    {
        result += await channels[i].ReadDoubleAsync();
    }

    Console.WriteLine("Result found: res = {0}, time = {1}", result, Math.Round((DateTime.Now - time).TotalSeconds, 3));

    var bytes = Encoding.UTF8.GetBytes(result.ToString());
    await moduleInfo.OutputWriter.WriteToFileAsync(bytes, moduleOptions.OutputFilename);

    for (int i = 0; i < pointsNumber; ++i)
    {
        await points[i].DeleteAsync();
    }
}
```

1

2

3

4

Рисунок 11 – Приклад АМ в PARCS-NET-K8

2.9. Огляд комунікації між точками

Подібно до PARCS-WCF, де було введено поняття локальної точки, дані в межах КП PARCS-NET-K8 передаються або мережею, або з використанням ITC. У першому випадку мережевим протоколом слугує TCP. Він забезпечує надійність та послідовність комунікації, використовує відносно небагато системних ресурсів та має широку підтримку засобів безпеки, таких як шифрування та аутентифікація (більш детально аргументи на користь його залучення в ПАРКС описано в [1], [2] та [9]).

Втім, необхідно також окреслити один доволі суттєвий недолік цього способу обміну інформацією: TCP нехтує границями повідомлень [38]. Іншими словами, серія викликів низькорівневого методу Send на TCP сокеті клієнта не обов'язково буде прочитана кожним окремим викликом методу Receive на TCP сокеті сервера. Така поведінка спричинена тим, що дані, отримувані шляхом виклику методу Receive, надходять не напряму з мережі. Насправді, вони беруться з внутрішнього TCP буфера, в який нові мережеві пакети послідовно складаються. Receive же зчитує усю наявну в буфері інформацію, а не лише перший пакет і тим самим можливо повертає ініціатору виклику більше або менше байтів, аніж той очікував отримати (див. рисунок 12).

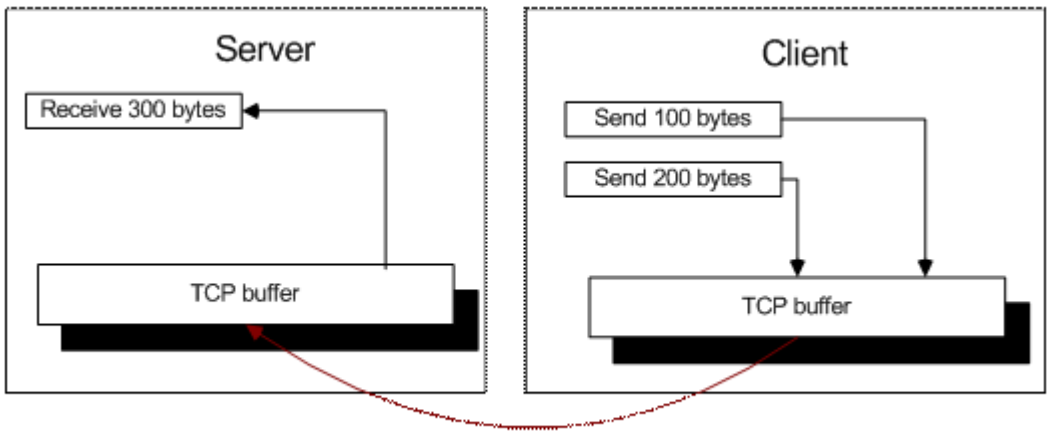


Рисунок 12 – TCP: два виклики Send зчитуються за один Receive

На щастя, у цієї проблеми границь TCP повідомлень є три можливих рішення: завжди надсилати повідомлення фіксованого розміру, надсилати розмір повідомлення перед самим повідомленням, або помічати початок та кінець кожного наступного повідомлення спеціальною послідовністю. Оскільки перший варіант накладає серйозні обмеження на вміст або форму даних, що передаються мережею, а третій вимагає побітової обробки кожного пакету в пошуку маркер-послідовностей, перевагу було віддано надсиланню очікуваного розміру. Таким чином, після надходження такого розміру точка-отримувач циклічно зчитує TCP буфер, допоки не набере заявлену кількість байтів, з чого сформує цілісне повідомлення.

Вище було, серед іншого, зазначено міжпотокову комунікацію. Справді, якщо фізично цільова точка знаходиться на цій же машині, підтримування витратного мережевого підключення є неоптимальним, що було показано в [9]. Зважаючи на це, в PARCS-NET-K8, як і в PARCS-WCF, було також додано локальний резервний спосіб обміну даних. На відміну від [2], де для цього було використано специфічну для WCF прив’язку NamedPipes, тут з цією метою експлуатується вбудований в .NET Core клас System.Threading.Channel. Відповідно до логіки, при створенні нової точки виконується розкриття цільової адреси та перевірка її на співпадіння з адресою поточного середовища. Якщо виявляється, що точка має бути створена на тому ж вузлі, виділяються два окремі канали (System.Threading.Channel) для локальної комунікації з цією точкою. Ідентифікатор каналів надсилається на внутрішній сервер, який згодом встановлює відповідне підключення та виконує ту ж обробку, що й сервер TCP.

2.10. Огляд контейнеризації та оркестрації

Піднята в [6] тема використання засобів контейнеризації в ПАРКС отримала новий виток в PARCS-NET-K8. Насамперед, цей напрям розвитку було визначено як пріоритетний через бажаний високий ступінь автоматизації на виході, чому використання такого легковагового, самодостатнього та незалежного від середовища рішення суттєво сприяє. Справді, як порівняти з

віртуальними машинами, контейнери є значно меншими за рахунок відсутності додаткових операційних систем [3939]. Це дозволяє розгортати їх набагато швидше та більше на одному фізичному сервісі. Окрім того, контейнери надають можливість запаковувати застосунки разом з їх залежностями, через що такі додатки працюють однаково стабільно як локально, так і в продуктовій середі.

Відсутність додаткового прошарку гостьових операційних систем, однак, має й свої недоліки. По-перше, контейнерам небажано надавати адміністративні повноваження, оскільки технологія контейнеризації має доступ до підсистем процесора, а тому наявність вразливості в одному з сервісів може призвести до компрометації усієї платформи. По-друге, побудова образів контейнерів має на увазі конкретну операційну систему: Windows або Linux. Остання вважається більш легковаговою та ефективною, адже споживає менше системних ресурсів. На жаль, донині запуск ПАРКС для .NET на Linux був неможливим через залежність попередніх реалізацій від операційної системи Windows. Втім, з появою багатоплатформності в PARCS-NET-K8 цей бар'єр було подолано.

На рисунку 13 зображено приклад Docker-файлу - інструкцій для створення Docker-образу демона з PARCS-NET-K8. Як можна побачити, збірка є багатоступеневою (з англ. «multistage build»). Спершу, на етапі base середовище виконання .NET визначається як базовий образ (рядки 3-4). На наступній стадії build в робочу директорію копіюється сирцевий код проектів Parcs.Daemon та Parcs.Core. Їх залежності розв'язуються, а код компілюється в режимі Release (рядки 6-13). Далі – стадія publish, в межах якої зібраний застосунок готується до публікації (рядки 15-16). Нарешті, останнім етапом, final, результуючі бінарні файли копіюються в основну директорію, після чого точкою входу встановлюється стандартна команда «dotnet Parcs.Daemon.dll» (рядки 18-21). Відзначимо, що базовий *mcr.microsoft.com/dotnet/runtime:7.0* містить тільки виконуваний код і бібліотеки, необхідні для роботи програми. Водночас, *mcr.microsoft.com/dotnet/sdk:7.0*, що використовується для компіляції та публікації є повноцінним набором засобів для розробки програмного забезпечення (з англ. «Software Development Kit»). Розділення процесу побудови

на стадії демонструє найголовнішу перевагу багатоступеневих збірок, а саме зменшення кінцевого розміру образу та, відповідно, його пришвидшення.

```
3   FROM mcr.microsoft.com/dotnet/runtime:7.0 AS base
4   WORKDIR /app
5
6   FROM mcr.microsoft.com/dotnet/sdk:7.0 AS build
7   WORKDIR /src
8   COPY ["src/Parcs.Daemon/Parcs.Daemon.csproj", "src/Parcs.Daemon/"]
9   COPY ["src/Parcs.Core/Parcs.Core.csproj", "src/Parcs.Core/"]
10  RUN dotnet restore "src/Parcs.Daemon/Parcs.Daemon.csproj"
11  COPY . .
12  WORKDIR "/src/src/Parcs.Daemon"
13  RUN dotnet build "Parcs.Daemon.csproj" -c Release -o /app/build
14
15  FROM build AS publish
16  RUN dotnet publish "Parcs.Daemon.csproj" -c Release -o /app/publish /p:UseAppHost=false
17
18  FROM base AS final
19  WORKDIR /app
20  COPY --from=publish /app/publish .
21  ENTRYPOINT ["dotnet", "Parcs.Daemon.dll"]
```

Рисунок 13 – Dockerfile з інструкціями для створення Docker-образу демона

Аналогічним до наведеного вище способом загортаються в контейнери і інші частини рішення: хост та портал. Для того ж, щоб ці основні компоненти разом з допоміжними сформували єдину функціональну систему, необхідно забезпечити їх взаємодію: щонайменше, порталу потрібно вміти спілкуватись з хостом, а хосту – з демоном. Для цього можна було б відкрити порти контейнерів для доступу ззовні, після чого разом з адресами публікації контейнерів прописати їх в конфігурації кожного застосунку. Однак, цей підхід не вважається загальноприйнятим, адже має низку недоліків. По-перше, прямий публічний доступ до окремих компонент може бути небажаним з точки зору безпеки: збільшується кількість точок входу в систему, а з ними – потенційна поверхня атаки (з англ. «attack surface»). По-друге, за таких умов між портами можуть виникати конфлікти на рівні мережевого інтерфейсу: якщо два процеси будуть очікувати на надходження пакетів через один і той самий порт, результат їх обробки буде непередбачуваним. По-третє, явне посилання одного контейнера на інший перешкоджає масштабуванню, адже кожен наступний доданий екземпляр має бути явно прописаний в конфігурації сервісу.

Через ці та інші причини натомість рекомендується використовувати засоби оркестрації контейнерів, які забирають на себе більшу частину питань з управління. З-поміж іншого, вони забезпечують балансування навантаження, адже механізми оркестрації можуть, керуючись внутрішніми евристиками утилізації ресурсів, автоматично розподіляти трафік між різними контейнерами. Окрім того, замість явних посилань на порти та адреси оркестрація пропонує розкриття сервісів, що дозволяє контейнерам автоматично знаходити одне одного та спілкуватись. Можна також виділити спрощену процедуру масштабування (замість ручного додавання нових екземплярів користувач лиш вказує бажану їх кількість), моніторинг стану контейнерів з їх автоматичним відновленням (здоров'я контейнерів постійно перевіряється і якщо якийсь з них виходить з ладу, засоби оркестрації миттєво створюють новий екземпляр), а також поетапні оновлення (нові сервіси публікуються поступово, контейнер за контейнером, таким чином особливо не перериваючи роботу представленого ними сервісу).

Серед інструментів оркестрації контейнерів вирізняють Kubernetes, Docker Compose, Mesos та інші. Docker Compose є порівняно нескладним у використанні та потребує найменших знань для старту роботи з ним: завдяки інтеграції з Visual Studio для побудови першого повноцінного багатоконтейнерного застосунку на його основі достатньо обрати проекти з підтримкою Docker, вказати які з них розкривають які порти і, за потреби, додати змінні середовища. На виході отримуємо YAML файл з декларацією контейнеризованої системи та, опціонально, допоміжний файл з налаштуваннями, що варіюються в залежності від середовища. Запустити цю систему можна простою командою «docker compose up», зупинити – «docker compose down». Якщо необхідно мати більше ніж один екземпляр сервісу, в YAML файлі вказується режим публікації «replicated», що автоматично запускає Docker Compose в конфігурації Swarm.

Саме Docker Compose, що автоматично постачається разом із утилітою Docker Desktop, було використано під час локальної розробки та відлагодження.

Втім, для продуктової середи та більш складних сценаріїв цей інструмент не так добре підходить, як, наприклад, Kubernetes.

Створений компанією Google Kubernetes є де-факто стандартом індустрії в сфері оркестрації контейнерів. Технологія має велику та активну спільноту, вичерпну документацію, є легко розширюваним та портативним PaaS рішенням, що підтримує як навантаження з наявністю стану, так і без нього. Окрім того, Kubernetes дозволяє накладати ресурсні ліміти та автоматично масштабувати сервіси. А завдяки відкритому коду та агностичній до середовища конфігурації відсутня прив'язка до конкретної операційної системи або будь-якого хмарного поставника послуг: Kubernetes може бути опублікований де завгодно.

Як і Docker Compose, в першу чергу середовище Kubernetes налаштовується декларативно. Замість послідовного виконання низки команд визначається один або декілька YAML файлів, що містять бажаний стан кластеру. Задача Kubernetes – зробити так, щоб опублікована система в будь-який момент часу відповідала цьому стану, незалежно від помилок та відмов. Платформою надається й імперативний інструмент, однак здебільшого ним користуються лише в крайових випадках, адже головною перевагою Kubernetes є саме здатність підтримувати систему в задекларованому стані.

На рисунку 14 наведено часткову конфігурацію Kubernetes для кластера PARCS-NET-K8, зокрема компоненти демона. Прямокутниками «1» та «4» тут означено типи двох публікації, Service та Deployment, які необхідні для його налаштування.

Service в Kubernetes - це абстракція, яка зазвичай надає статичну IP-адресу та DNS-ім'я для групи контейнерів, що виконують один і той же застосунок. Вона дозволяє іншим додаткам звертатися до цих контейнерів незалежно від того, на яких фізичних машинах вони розташовані. Зазначимо також, що продемонстрована конфігурація визначає спеціальний тип сервісу, безголовий (з англ. «headless»), на що вказує відмічене третім прямокутником значення «none» параметра clusterIP. За такої конфігурації сервісу parcs-daemon-headless не буде надано єдину статичну адресу, за якою можна було б звернутись до демонів.

Натомість, запит на розв’язання такої адреси за закріпленим DNS іменем повертатиме адреси усіх контейнерів, на котрих виконується демон-програма. Таким чином реалізовано динамічне розкриття демонів в середовищі Kubernetes: шляхом надсилання DNS запиту хост отримує змогу в моменті динамічно визначати місцезнаходження доступних для виконання обчислень процесів.

Deployment - це об'єкт Kubernetes, який безпосередньо описує сам додаток та його потреби в ресурсах. Deployment контролює кількість реплік додатку, що запущені в кластері, та забезпечує відповідність цієї кількості задекларованому числу. Відповідно до даної специфікації, розгортається три контейнери (див. прямокутник «5» на рисунку 14) на основі образу oleksiibohusevych/parcsdaemon з реєстру за замовчуванням Docker Hub (див. прямокутники «6» та «7» на рисунку 14). Додатково, прямокутником «8» позначено монтування виділеного середовищем сховища в файлову систему контейнерів, а прямокутниками «2» та «7» - зв’язування портів демон-контейнерів та демон-сервісу.

Рисунок 14 – Декларація компонент демона в середовищі Kubernetes

2.11. Огляд схеми публікації

Як вже було сказано, Kubernetes є агностичною до середовища технологією, тож може бути однаково ефективно розміщений як на виділеному сервері, так і в будь-якій популярній хмарі. Перший варіант надає більший ступінь контролю (програмне забезпечення є повністю закритим у фізично доступній апаратній середі). Хмарна ж опція традиційно має низку переваг, серед яких економічна ефективність (використовується рівно стільки ресурсів, скільки вимагає попит, тож немає простою), доступність та миттєве масштабування (тисячі повністю налаштованих машин по всьому світу завжди готові задовольнити будь-які обчислювальні потреби), просунута безпека (хмарні провайдери зазвичай пропонують екстенсивні механізми захисту), відсутність потреби в постійному обслуговувані (інфраструктурою управляє постачальник) тощо. Серед готових хмарних рішень Kubernetes найбільш відомими є Azure Kubernetes Service від Microsoft, Kubernetes on AWS від Amazon та Google Kubernetes Engine від Google.

У кінцевому підсумку вибір цільової платформи залежить від багатьох факторів, як-от від дисконтних домовленостей з конкретним провайдером, ціни за обчислювальні ресурси, відмінностей в запропонованому функціоналі, інтегрованості з існуючими рішеннями, корпоративними політиками тощо. В розрізі ж PARCS-NET-K8, фінальне місце призначення публікації не є особливо критичним, адже відтворення та проектування системи на будь-яке інше середовище за умови наявності декларативних YAML файлів не становить труднощів. З огляду на принцип сумісності технологій, незначну перевагу можна віддати Azure Kubernetes Service (AKS), оскільки, так само як і платформа .NET, зазначений сервіс належить до сімейства Microsoft.

Безпосередньо створити ресурс AKS на хмарній платформі Azure можна багатьма способами. Найпростіший – вручну, через портал системи. Для цього у вкладці створення ресурсів необхідно знайти однойменний, вказати ім’я, версію Kubernetes, регіон, політику масштабування, категорію віртуальних машин. За потреби звідти ж можна налаштувати логування, автентифікацію, моніторинг,

розмір пулу, інтеграцію з приватними реєстрами образів тощо. Після створення ресурсу на вкладці з інформацією про нього можна в декілька кроків виконати публікацію YAML файлу (див. рисунок 15), знайти всі наявні сервіси, ознайомитись зі станом інфраструктури, перевірити логування тощо.

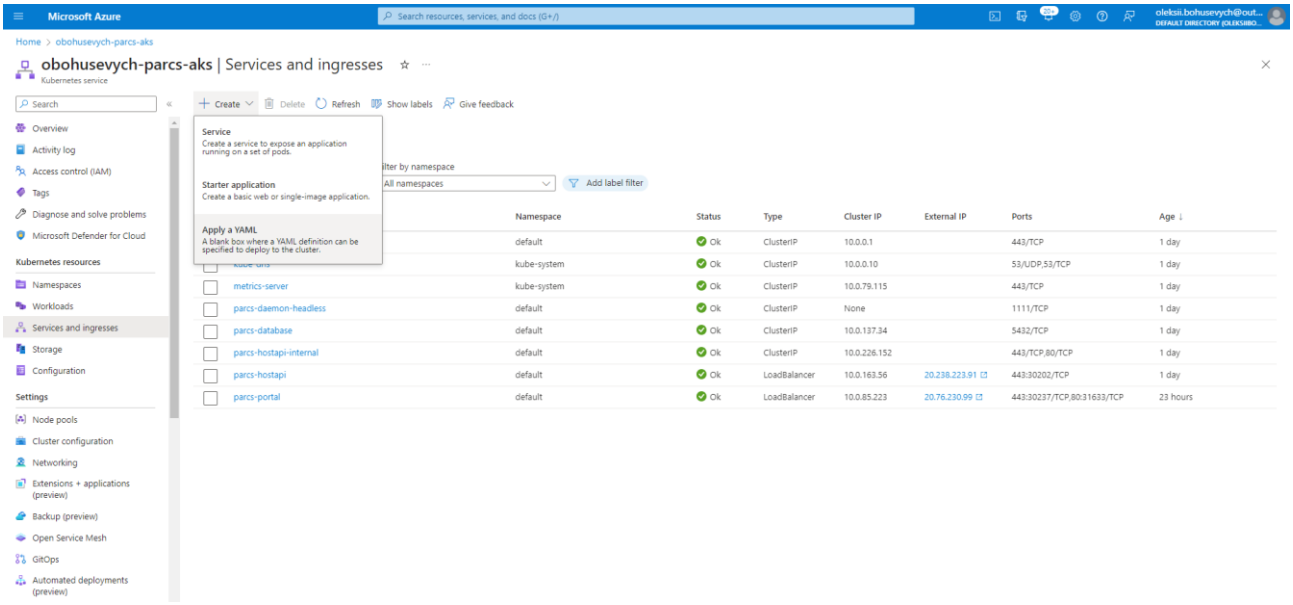


Рисунок 15 – AKS кластер PARCS: панель сервісів, публікація YAML

Аналогічний ефект досягається шляхом взаємодії з внутрішнім для Azure програмним інтерфейсом – Azure Resource Manager (ARM). Одним зі способів такої взаємодії є створення доволі громіздких ARM шаблонів – декларативного представлення ресурсів Azure у вигляді форматованих JSON файлів. Іншим варіантом є написання імперативних скриптів з використанням утиліти командного рядку Azure CLI або фреймворку автоматизації Azure PowerShell.

Втім, останнім часом зазначені вище підходи доволі небезпідставно програють конкуренцію новим засобам категорії Infrastructure as a Code (IaaC), до яких, наприклад, належать Terraform від HashiCorp та Bicep від Microsoft. Ці засоби, як випливає з назви, дозволяють розглядати інфраструктуру як код, що привносить такі зручності як система контролю версій, модульність та декларативність. Terraform є дещо більш просунутою технологією з підтримкою планування, відкочуванням змін, збереженням стану та націленістю на низку популярних хмарних платформ (AWS, Azure, GCP, OCI). Водночас, Bicep - це,

по суті, високорівневий інтерпретатор для спрощення написання ARM шаблонів шляхом впровадження специфічного для домену синтаксису.

Для того, щоб полегшити процес публікації та початок роботи з PARCS-NET-K8, було вирішено створити ARM шаблон за допомогою Bicep. На відміну від більш комплексного варіанту з Terraform, в цьому випадку від користувача не вимагається ні встановлення залежностей утиліти, ні знання про особливості роботи з нею (для розгортання ресурсів через Terraform необхідно виконати terraform apply, після чого погодитись із запропонованим планом публікації). Таким чином, його дії зводяться до завантаження єдиного файлу, ARM шаблону, та створення ресурсу «Публікація ARM шаблону» на основі цього файлу.

2.12. Огляд можливостей масштабування

Підкріплена просунутими функціями платформ Kubernetes та Azure, система PARCS-NET-K8 забезпечує екстенсивність горизонтального й вертикального масштабування. Можливим є як виконання невибагливих обчислень на локальному комп'ютері з одним ядром, так і розв'язання трудомістких промислових задач за рахунок ферм надпотужних процесорів.

Першим рівнем розширення тут є програмне збільшення щільності локальних точок керуючого простору, тобто потоків. Другим рівнем є розгортання додаткових демонів, тобто процесів, шляхом зміни значення атрибуту replicas в декларації кластеру. Як тільки контейнер нового демона успішно запущено, за іменем безголового сервісу на DNS сервері кластеру додаються відповідні записи типу A та AAAA. Відтак, DNS запити проти цього сервісу автоматично повертатимуть адресу нового демона. На останок, третім рівнем розширення є обчислювальні вузли, тобто віртуальні машини, додавання яких є питанням зміни параметра на панелі пулу кластерних вузлів. Після розміщення ж машин Kubernetes збалансує та розподілить нові й існуючі навантаження, вже враховуючи створені ресурси. В разі потреби є також опція автоматичного масштабування, за якого вбудований в AKS сервіс створюватиме або знищуватиме контейнери та вузли зважаючи на поточне навантаження.

РОЗДІЛ 3. ТЕСТУВАННЯ PARCS-NET-K8

Тестування системи PARCS-NET-K8 проводилось в середовищі AKS, керованому рішенні Kubernetes від хмарної платформи Microsoft Azure. Усі наведені в цьому розділі алгоритмічні модулі було зареєстровано в системі в конфігурації Release, яка містить усі можливі оптимізації рівня застосунку. Результати тестування варіюються в залежності від кількості та щільності точок, числа демонів, пулу вузлів, а також апаратної конфігурації. Розмір машин тут було зафіксовано для зменшення обсягів роботи. Ним обрано рівень Standard_DS2_v2, що дає 2 віртуальних ядра та 7 гігабайт оперативної пам’яті. За потреби можна провести додаткові заміри в залежності від потужності машин.

3.1. Задача доказу виконаної роботи

Доказ виконаної роботи (з англ «Proof of work») – це базовий концепт криптографічної системи Bitcoin, а також інших програмних середовищ, в яких необхідно швидко й однозначно надавати підтвердження виконання складних та довготривалих обчислень.

Один з підходів до реалізації цього концепту базується на односторонніх функціях хешування (напр. SHA-256): клієнт передає обробнику довільний текст та число, після чого останній повинен знайти таке доповнення до заданого тексту, щоб кількість нулів на початку хешу від них двох дорівнювала заданому числу. Зрозуміло, в такій постановці обробнику залишається лиш перебирати довільні комбінації, адже непередбачуваність результату застосування хеш-функції робить можливим отримання задовільного значення в будь-який момент. При цьому, як тільки доповнення знайдено, ініціатор може миттєво пересвідчитись в його коректності: для цього достатньо лиш одного обчислення значення хеш-функції на результаті та початковому тексті.

Власне робота, тобто довільний пошук в нескінченній множині доповнень, може бути виконана паралельно. Для цього увесь простір можливих значень необхідно розбити на фіксовані частини, які згодом надати кожному вузлу на обробку. Процес продовжується допоки один із виконавців не отримає

задовільний результат. Після цього всі поточні операції скасовується, а шукане значення повертається ініціатору виклику. Саме за таким принципом створено паралельну реалізацію цієї задачі засобами системи PARCS-NET-K8. Результати тестування паралельної й послідовної реалізації наведено в таблиці 1.

N	Послідовний	Паралельний			
		2 вузли		4 вузли	
		Час (с.)	Прискорення (%)	Час (с.)	Прискорення (%)
5	0.210	0.75934	-261.59	0.36529	-73.94
6	85.939	71.24072	17.10	51.48883	40.08
7	445.27	405.99072	8.82	251.66452	43.48
8	11362.01	7848.09502	30.92	3961.45211	65.13

Таблиця 1 – Час виконання алгоритмічного модуля «Доказ роботи»; тут «N» – складність; к-сть точок – 10, к-сть демонів дорівнює кількості вузлів, розмір пакету для перевірки – 100000

3.2. Задача множення матриць

Результатом множення матриці A розміру m×n на матрицю B розміром n×1 є матриця C розміру m×1, де кожний елемент матриці C визначається наступним рівнянням:

$$c_{ij} = \sum_{k=0}^{n-1} a_{ik} \cdot b_{kj}, 0 \le i < m, 0 \le j < l$$

Як можна побачити, кожен елемент матриці C є скалярним добутком відповідного рядка матриці A та стовпчика матриці B:

$$c_{ij} = \left(a_i, b_j^T\right), a_i = \left(a_{i0}, a_{i1}, \ldots, a_{in-1}\right), b_j^T = \left(b_{0j}, b_{1j}, \ldots, b_{n-1j}\right)^T$$

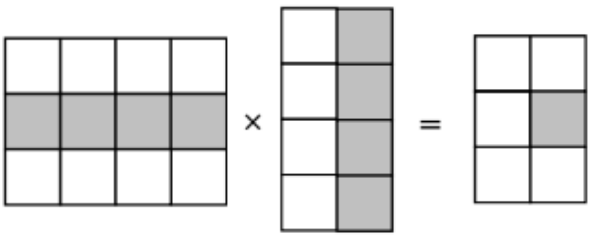


Рисунок 16 – Графічне представлення процесу множення двох матриць

Таким чином, наприклад, якщо матрицю A, що складається з 3 рядків та 4 стовпчиків, помножити на матрицю B, що складається з 4 рядків та 2 стовпчиків, буде отримано матрицю C, що складається з 3 рядків та 3 стовпчиків:

$$\begin{pmatrix} 3 & 2 & 0 & -1 \\ 5 & -2 & 1 & 1 \\ 1 & 0 & -1 & -1 \end{pmatrix} \times \begin{pmatrix} 1 & -1 \\ 2 & 5 \\ -3 & 2 \\ 7 & 4 \end{pmatrix} = \begin{pmatrix} 0 & 3 \\ 5 & -9 \\ -3 & -7 \end{pmatrix}$$

В базовому варіанті для вирахування матриці C при довільних A і B з відповідними розмірами m×n та n×k необхідно виконати m×k операцій множення рядка на стовпчик, кожна з яких включає поелементне множення окремих елементів та складання отриманих добутків. Процес можна значно пришвидшити за рахунок формули блочного обчислення добутку матриць, як це було запропоновано в [40]:

$$A = \begin{pmatrix} A_{11} A_{12} \\ A_{21} A_{22} \end{pmatrix}$$

$$B = \begin{pmatrix} B_{11} B_{12} \\ B_{21} B_{22} \end{pmatrix}$$

$$AB = \begin{pmatrix} A_{11} * B_{11} + A_{12} * B_{21} & A_{11} * B_{12} + A_{12} * B_{22} \\ A_{21} * B_{11} + A_{22} * B_{21} & A_{21} * B_{12} + A_{22} * B_{22} \end{pmatrix}$$

Відповідно до алгоритму, який і було використано під час розв’язання даної задачі, рекурсивно скорочуючи розмірність матриць, початкові A і B можна розбити на скільки завгодно малі частини, кожну з яких згодом виконати незалежно. Тут варто також зазначити, що чим глибшим росте дерево рекурсії, тим більші й відповідні накладні витрати. Тож необов’язково, що трикратне розбиття буде швидшим за двократне. Скоріше навпаки. Результати застосування такого підходу при побудові відповідного алгоритмічного модуля в системі PARCS-NET-K8 наведено в таблиці 2.

N	Послідовний	Паралельний			
		2 вузли		4 вузли	
		Час (с.)	Прискорення (%)	Час (с.)	Прискорення (%)
512	2.77732	1.79784	35.26	1.12365	59.54
1024	23.58286	15.31127	35.07	8.99900	61.84
1536	95.52184	62.32363	34.75	32.09683	66.39
2048	311.82199	193.39207	37.97	112.39377	63.95
2560	764.22659	482.80361	36.82	209.17263	72.62
3072	975.10869	648.89927	33.45	312.00008	68.00

Таблиця 2 – Час виконання послідовної та паралельної версії алгоритмічного модуля «Множення матриць»; тут «N» – розмір матриці (N×N); к-сть точок відповідає обраному рівню рекурсії (2), помноженому на число блоків (8)

3.2. Задача про найкоротший шлях

Нехай дано зважений орієнтовний граф (див. рисунок 17) із додатними або від’ємними вагами ребер. Для кожної пари вершин необхідно знайти такий шлях між ними, що сума зустрітих на ньому ваг буде мінімальною.

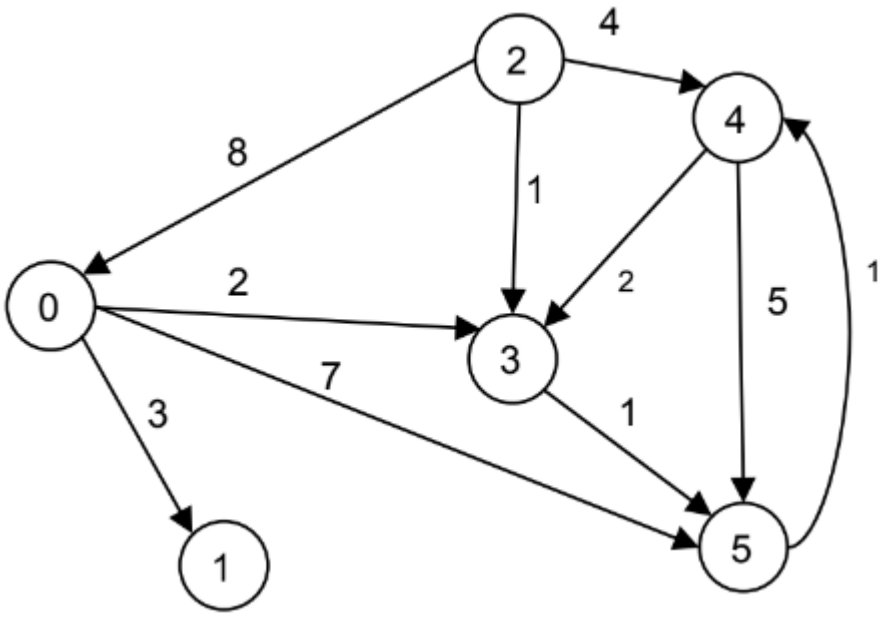


Рисунок 17 – Приклад зваженого орієнтованого графа

Для розв’язання цієї задачі зазвичай використовується алгоритм Флойда-Воршелла, послідовна версія якого може бути описана наступним чином:

```
1.  for (k = 0; k < n; k++)
2.      for (i = 0; i < n; i++)
3.          for (j = 0; j < n; j++)
4.              A[i,j] = min(A[i,j],A[i,k]+A[k,j]);
```

Тут A – квадратна матриця суміжності графа розміром n×n. Як можна побачити, основна ідея алгоритму полягає в тому, що обчислюються і порівнюються між собою усі можливі відстані між парами вершин. Відповідно, складність тут – $O(n^3)$, де n – кількість вершин графа.

Паралельна версія Флойда-Воршелла ґрунтується на розбитті матриці суміжності на горизонтальні або вертикальні блоки та часткове незалежне обчислення відстаней в них. Оскільки кожна така підзадача має також обробляти вершини поза власним блоком (див. рисунок 18), впродовж виконання алгоритму постійно відбувається обмін чужими рядками між процесами.

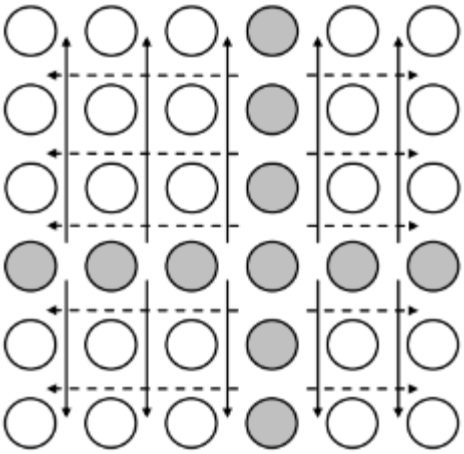


Рисунок 18 – Інформаційні залежності в алгоритмі Флойда-Воршелла

N	Послідовний	Паралельний			
		2 вузли		4 вузли	
		Час (с.)	Прискорення (%)	Час (с.)	Прискорення (%)
1000	5,23541	3,67211	29.86	2,42921	53.60
2000	41,22188	21,35144	48.20	10,60155	74.28
3000	140,53901	68,85207	51.00	36,31832	74.15
4000	333,42421	162,61011	51.23	85,42581	74.37
5000	648,84955	316,65243	51.19	167,75119	74.14

Таблиця 3 – Час виконання послідовної та паралельної версії алгоритмічного модуля «Пошук найкоротшого шляху»; тут «N» – потужність множини вершин графу; к-сть точок дорівнює к-сті вузлів та демонів

ВИСНОВКИ

Розглянуто та проаналізовано попередні роботи в предметній області. Керуючись сучасними підходами в розробці програмного забезпечення та напрацюваннями в сфері, реалізовано систему ПАРКС на базі сучасної версії платформи .NET та технології оркестрації контейнерів Kubernetes. Запропоновано модель публікації рішення на хмарну платформу, створено засоби автоматизованого розгортання. Додатково, проведено тестування системи на прикладах загальних паралельно-рекурсивних алгоритмів.

Перспективами розвитку роботи вважається налаштування механізмів автентифікації та авторизації, глибша інтеграція з моніторинговими системами для покращення досвіду відлагодження застосунків, впровадження контролю доступу та більшої ізоляції алгоритмічних модулів, оновлення компонент до .NET 8, наступної версії платформи з приміткою LTS, а також публікація системи в інші популярні хмарні середовища.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Дерев'янченко О. В. СИСТЕМА ПАРАЛЕЛЬНИХ ОБЧИСЛЕНЬ НА КОМП'ЮТЕРНІЙ МЕРЕЖІ ПАРКС-.NET / О. В. Деревянченко, А. Ю. Хавро. – Тернопіль: Київський національний університет імені Тараса Шевченка, 2014.

2. Федорус О. М. ЗАСТОСУВАННЯ СИСТЕМ ПАРКС ДЛЯ МОДЕЛЮВАННЯ ОПЕРАЦІЙ РЕЛЯЦІЙНОЇ АЛГЕБРИ ВИБОРУ: дис. канд. техн. наук / Федорус Олексій Мстиславович – Київ: Київський національний університет імені Тараса Шевченка, Україна, 2021. – 123 с.

3. What Is Moore's Law and Is It Still Relevant in 2022? [Електронний ресурс] // TINA SIEBER. – 2022. – Режим доступу до ресурсу: https://www.makeuseof.com/tag/what-is-moores-law-and-what-does-it-have-to-do-with-you-makeuseof-explains/.

4. Lock S. What is AI chatbot phenomenon ChatGPT and could it replace humans? [Електронний ресурс] / Samantha Lock // The Guardian. – 2022. – Режим доступу до ресурсу: https://amp.theguardian.com/technology/2022/dec/05/what-is-ai-chatbot-phenomenon-chatgpt-and-could-it-replace-humans.

5. Parallel Computing and Its Modern Uses [Електронний ресурс] // HP TECH TAKES. – 2019. – Режим доступу до ресурсу: https://www.hp.com/us-en/shop/tech-takes/parallel-computing-and-its-modern-uses.

6. Анісімов А. В. ЗАСТОСУВАННЯ СИСТЕМИ ПАРКС.NET В DOCKER КОНТЕЙНЕРАХ ТА GOOGLE CLOUD PLATFORM ДЛЯ РОЗПОДІЛЕНИХ ХМАРНИХ ОБЧИСЛЕНЬ / А. В. Анісімов, А. Ю. Хавро. – Київ: Київський національний університет імені Тараса Шевченка, Україна, 2018. – 10 с.

7. Навчальні плани [Електронний ресурс] // Київський національний університет імені Тараса Шевченка Факультет комп'ютерних наук та кібернетики. – 2023. – Режим доступу до ресурсу: https://csc.knu.ua/uk/curriculum.

8. Федорус О. М. ПАРКС ЯК ЗАСІБ РЕАЛІЗАЦІЇ РОЗПОДІЛЕНИХ ХМАРНИХ ОБЧИСЛЕНЬ / Олексій Мстиславович Федорус. – Київ: Київський

національний університет ім. Тараса Шевченка факультет кібернетики, 2016. – 6 с.

9. Дерев'янченко О. В. ПАРКС-JAVA система для паралельних обчислень на комп'ютерних мережах / Олександр Валерійович Дерев'янченко. – Київ: Київський національний університет імені Тараса Шевченка, Україна, 2011. – 60 с.

10. Анисимов А. В. ТЕХНОЛОГИЯ ПАРАЛЛЕЛЬНОГО ПРОГРАММИРОВАНИЯ «ПАРУС» / А. В. Анисимов, Ю. Е. Борейша, П. П. Кулябко. – Киев: Киевский государственный университет, 1991. – 9 с.

11. Анисимов А. В. Разработка и анализ архитектуры распределенной системы программирования ПАРУС / Анатолий Васильевич Анисимов. – Киев: ВИНИТИ, 1982. – 1 с.

12. Parcs: Core library for Parcs.NET - Parallel distributed system for .NET. [Електронний ресурс] // AndriyKhavro. – 2017. – Режим доступу до ресурсу: https://www.nuget.org/packages/Parcs.Module.CommandLine.

13. Parcs.Module.CommandLine: Core library for Parcs.NET [Електронний ресурс] // AndriyKhavro. – 2017. – Режим доступу до ресурсу: https://www.nuget.org/packages/Parcs.Module.CommandLine.

14. Дерев'янченко О.В., Хавро А.Ю. Застосування системи ПАРКС.NET та Amazon EC2 для хмарних обчислень // Вісник Київського національного університету імені Тараса Шевченка Серія фізико-математичні науки, 2015, №4, С.111-118.

15. How to: PARCS.NET on Azure [Електронний ресурс] // AndriyKhavro. – 2019. – Режим доступу до ресурсу: http://parcs.unicyb.kiev.ua/mr/PARCS.NET_AWS.pdf.

16. Microsoft .NET Framework Lifecycle [Електронний ресурс] // Microsoft. – 2023. – Режим доступу до ресурсу: https://learn.microsoft.com/en-us/lifecycle/products/microsoft-net-framework.

17. CALLAN D. .NET Framework 4.8 v .NET 7 reflection performance benchmarks [Електронний ресурс] / DAVE CALLAN // Dave Callan's .NET Blog. –

2023. – Режим доступу до ресурсу: https://davecallan.com/dotnet-framework-48-v-dotnet7-reflection-performance-benchmarks/.

18. Shared responsibility in the cloud [Електронний ресурс] // Microsoft. – 2022. – Режим доступу до ресурсу: https://learn.microsoft.com/en-us/azure/security/fundamentals/shared-responsibility.

19. Parcs.NET [Електронний ресурс] // AndriyKhavro. – 2018. – Режим доступу до ресурсу: https://github.com/AndriyKhavro/Parcs.NET.

20. Анисимов А.В., Кулябко П.П., Годованюк М.И. Паралельне програмування в мережах на осові ПАРКС-технології (базова мова Python) // Вісник Київського національного університету імені Тараса Шевченка, 2016, № 3, С. 57-60.

21. What is "managed code"? [Електронний ресурс] // 2021 – Режим доступу до ресурсу: https://learn.microsoft.com/en-us/dotnet/standard/managed-code.

22. Asynchronous programming with async and await [Електронний ресурс] // Microsoft. – 2023. – Режим доступу до ресурсу: https://learn.microsoft.com/en-us/dotnet/csharp/asynchronous-programming.

23. Host WCF Service in Azure App Service [Електронний ресурс] // AspDotNetCodeHelp. – 2017. – Режим доступу до ресурсу: https://aspdotnetcodehelp.wordpress.com/2017/11/13/hosting-wcf-in-azure-app-service/.

24. CHANDLER H. Microservices vs. monolithic architecture [Електронний ресурс] / HARRIS CHANDLER // Atlassian. – 2023. – Режим доступу до ресурсу: https://www.atlassian.com/microservices/microservices-architecture/microservices-vs-monolith.

25. Brown S. The C4 model for visualizing software architecture [Електронний ресурс] / Simon Brown. – 2018. – Режим доступу до ресурсу: https://c4model.com/.

26. .NET and .NET Core Support Policy [Електронний ресурс] // Microsoft. – 2023. – Режим доступу до ресурсу: https://dotnet.microsoft.com/en-us/platform/support/policy/dotnet-core.

27. Toub S. Performance Improvements in .NET 7 [Електронний ресурс] / Stephen Toub = // Mi. – 2022. – Режим доступу до ресурсу: https://devblogs.microsoft.com/dotnet/performance_improvements_in_net_7/#reflection.

28. Migrate from ASP.NET Core 6.0 to 7.0 [Електронний ресурс] // Microsoft. – 2022. – Режим доступу до ресурсу: https://learn.microsoft.com/en-us/aspnet/core/migration/60-70.

29. .NET vs. .NET Framework for server apps [Електронний ресурс] // Microsoft. – 2023. – Режим доступу до ресурсу: https://learn.microsoft.com/en-us/dotnet/standard/choosing-core-framework-server.

30. What's new in .NET 7 [Електронний ресурс] // Microsoft. – 2022. – Режим доступу до ресурсу: https://learn.microsoft.com/en-us/dotnet/core/whats-new/dotnet-7.

31. Tutorial: Create a minimal API with ASP.NET Core [Електронний ресурс] // Microsoft. – 2023. – Режим доступу до ресурсу: https://learn.microsoft.com/en-us/aspnet/core/tutorials/min-web-api.

32. Gillis A. S. REST API (RESTful API) [Електронний ресурс] / Alexander S. Gillis // TechTarget – Режим доступу до ресурсу: https://www.techtarget.com/searchapparchitecture/definition/RESTful-API.

33. Fowler M. Patterns of Enterprise Application Architecture / Martin Fowler., 2002. – 560 с. – (1st).

34. Toub S. An Introduction to System.Threading.Channels [Електронний ресурс] / Stephen Toub // MSFT. – 2019. – Режим доступу до ресурсу: https://devblogs.microsoft.com/dotnet/an-introduction-to-system-threading-channels/.

35. ASP.NET Core Blazor [Електронний ресурс] // Microsoft. – 2023. – Режим доступу до ресурсу: https://learn.microsoft.com/en-us/aspnet/core/blazor/?view=aspnetcore-7.0.

36. Parcs.Net [Електронний ресурс] // Oleksii Bohuseyvch. – 2023. – Режим доступу до ресурсу: https://www.nuget.org/packages/Parcs.Net/.

37. Create a .NET Core application with plugins [Електронний ресурс] // Microsoft. – 2022. – Режим доступу до ресурсу: https://learn.microsoft.com/en-us/dotnet/core/tutorials/creating-app-with-plugin-support.

38. Talazar A. Solution for TCP/IP client socket message boundary problem [Електронний ресурс] / Alex Talazar // codeproject. – 2005. – Режим доступу до ресурсу: https://www.codeproject.com/Articles/11922/Solution-for-TCP-IP-client-socket-message-boundary.

39. Docker vs. Virtual Machines: Differences You Should Know [Електронний ресурс] // 2022 – Режим доступу до ресурсу: https://cloudacademy.com/blog/docker-vs-virtual-machines-differences-you-should-know/.

40. Федорус О.М. Застосування ПАРКС для моделювання паралельнорекурсивних процесів // Матеріали Міжнародної наукової молодіжної школи «СИСТЕМИ ТА ЗАСОБИ ШТУЧНОГО ІНТЕЛЕКТУ» AIIS'2017, С. 215-218 117.