УДК 004.272.26 + 004.021 + 004.032.26 https://doi.org/10.15407/pp2021.03.003

І.З. Ашур, А.Ю. Дорошенко

РОЗПОДІЛЕНА РЕАЛІЗАЦІЯ МЕТОДУ НЕЙРОЕВОЛЮЦІЇ НАРОСТАЮЧОЇ ТОПОЛОГІЇ

У статті запропонована нова розподілена реалізація методу нейроеволюції наростаючої топології, яка, за наявності достатніх обчислювальних ресурсів, дозволяє радикально збільшити швидкість знаходження оптимальних конфігурацій нейронних мереж. З метою оптимізації продуктивності рішення, рівномірного розподілу завдань між вузлами та оптимального використання обчислювальних ресурсів була реалізована підтримка пакетної оцінки геномів. Експериментальна перевірка нової реалізації засвідчує, що, використовуючи запропоноване розподілене рішення, швидкість виконання методу нейроеволюції наростаючої топології в частині оцінювання згенерованих нейронних мереж на прикладі розглянутого завдання і середовища може зростати на декілька порядків.

Ключові слова: NEAT, нейроеволюція наростаючої топології, штучні нейронні мережі, навчання з підкріпленням, генетичні алгоритми, розподілені обчислення, хмарні обчислення.

Вступ

Попри сильні сторони методу нейроеволюції наростаючої топології, якот можливість його застосування в завданнях, де важко обрати функцію витрат і топологію нейронної мережі, однією з проблем нейроеволюції і, зокрема, методу нейроеволюції наростаючої топології, є повільна конвергенція до оптимальних результатів, особливо у випадку роботи з комплексними та складними середовищами. Ця робота пропонує розподілену реалізацію методу нейроеволюції наростаючої топології, що, за наявності достатніх обчислювальних ресурсів, дозволяє радикально збільшити швидкість знаходження оптимальних конфігурацій нейронних мереж.

Нейроеволюція

Нейроеволюція – форма машинного навчання, яка використовує еволюційні алгоритми для генерації штучних нейронних мереж, їхніх параметрів, топологій і правил. Головна перевага нейроеволюції полягає в можливості її ширшого застосування порівняно з навчанням з учителем, яке вимагає розмічених коректних пар вхідних та вихідних даних для тренування. На відміну від цього, нейроеволюція потребує лише можливості оцінити ефективність згенерованої мережі на будь-якому етапі навчання. Наприклад, результативність гіпотетичної ігрової партії у вигляді перемоги одного чи іншого гравця можна легко оцінити без надання розмічених даних про бажані або ефективні ігрові стратегії [1].

Нейроеволюцію часто розуміють як частину парадигми навчання з підкріпленням, що може бути протиставлена загальноприйнятим методам глибинного навчання, які використовують градієнтний спуск на штучних нейронних мережах із фіксованою топологією [2].

NEAT

Одне з головних питань нейроеволюції полягає у використанні переваг еволюції топології нейронної мережі поряд з еволюцією ваги з'єднань її вузлів. Нейроеволюція наростаючої топології (NeuroEvolution of Augmenting Topologies, NEAT) – генетичний алгоритм знаходження штучних нейронних мереж шляхом еволюції (нейроеволюційний метод), розроблений 2002 року дослідником Кеном Стенлі (Ken Stanley) [3].

Традиційно топологію нейронної мережі обирає людина-експериментатор, а значення ваги з'єднань між вузлами-нейронами отримують у процесі навчання. Це призводить до ситуації, коли необхідно застосувати підхід проб та помилок для того, аби знайти вдалу топологію для поставленого завдання. NEAT – приклад алгоритму, що шляхом еволюції одночас-

но змінює і топологію, і значення ваги з'єднань штучної нейронної мережі. Такі алгоритми належать до класу TWEANN (Topology and Weight Evolving Artificial Neural Network) [4].

Алгоритм нейроеволюції наростаючої топології починає процес еволюції зі штучної нейронної мережі, подібної до перцептрона, з лише вхідним та вихідним заздалегідь визначеними шарами вузлів-нейронів. Із плином дискретних кроків нейроеволюції складність топології нейронної мережі може зростати шляхом створення нових вузлів-нейронів у присутніх шляхах з'єднань або внаслідок створення нових з'єднань між попередньо роз'єднаними нейронами.

Метод нейроеволюції наростаючої топології перевершує найкращі методи нейроеволюції фіксованих топологій у завданнях навчання з підкріпленням. Нейроеволюція наростаючої топології демонструє високі показники ефективності завдяки використанню принципового методу кросинговеру різних топологій, захисту структурної інновації внаслідок видоутворення та інкрементального росту з мінімальних структур [5-7].

SharpNEAT

SharpNEAT – це .NET C# реалізація еволюційного алгоритму, а саме алгоритму еволюції нейронних мереж. Еволюційний алгоритм використовує еволюційні механізми мутації, генетичної рекомбінації та відбору для пошуку нейронних мереж, функціонал та поведінка котрих задовольнила би попередньо визначене завдання. Реалізація SharpNEAT розроблена дослідником Коліном Гріном (Colin Green) [8].

Алгоритм NEAT та реалізація SharpNEAT виконують пошук не тільки структури нейронної мережі, тобто її вузлів і з'єднань між ними, а й ваги цих з'єднань. Ця особливість вдало виділяє алгоритм нейроеволюції наростаючої топології з-поміж інших сучасних технік нейроеволюції та технік навчання з підкріпленням.

Фреймворк і демонстраційне програмне забезпечення SharpNEAT надає набір вбудованих задач-прикладів для демонстрації роботи алгоритму нейроеволюції. Реалізація SharpNEAT є модульною, що дозволяє легко вносити зміни, розширювати та повторно використовувати її частини. SharpNEAT створений, зокрема, для задоволення інтересів питань біологічної еволюції та границь можливостей нейроеволюції в зрізі рівня складності проблем, рішення котрих можуть запропонувати алгоритми нейроеволюції. На рис. 1 зображено високорівневу діаграму основних модулів оригінального проєкту SharpNEAT.

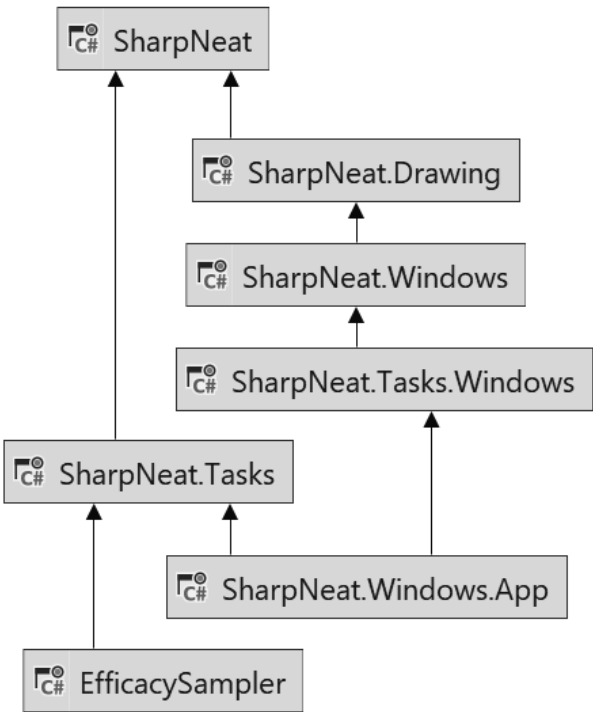

Рисунок 1. Високорівнева діаграма основних модулів оригінального проєкту SharpNEAT

- Модуль SharpNeat є базовим модулем із головними інтерфейсами й реалізацією функціоналу, пов'язаного зі штучними нейронними мережами та графами, функціями активації нейронів, оцінкою придатності геномів і феномів, алгоритмом нейроеволюції, експериментами, серіалізацією та десеріалізацією.

- Модуль SharpNeat.Tasks – модуль із вбудованими задачами навчання з підкріпленням.

- Модуль SharpNeat.Drawing відповідає за вбудований функціонал рендерингу штучних нейронних мереж.

- Модуль SharpNeat.Windows відповідає за високорівневі абстракції графічного інтерфейсу SharpNeat.

- Модуль SharpNeat.Windows.App відповідає за реалізацію графічного інтерфейсу SharpNeat для сімейства операційних систем Microsoft Windows.

- Модуль EfficacySampler відповідає за збір метрик ефективності алгоритму нейроеволюції наростаючої топології: часу виконання, кількості поколінь, показників придатності та складності тощо.

На рис. 2 зображено високорівневу блок-схему алгоритму нейроеволюції наростаючої топології в реалізації програмного фреймворку SharpNEAT.

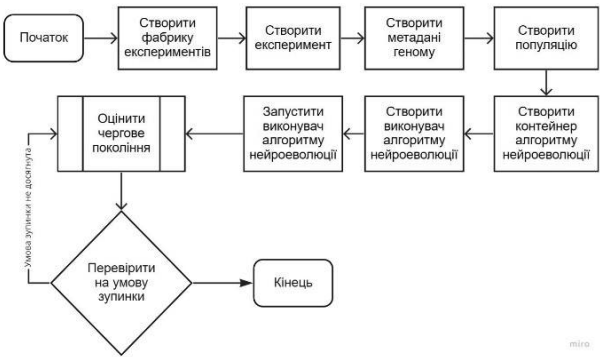

Рисунок 2. Високорівнева блок-схема алгоритму нейроеволюції наростаючої топології в реалізації програмного фреймворку SharpNEAT

Розподілені обчислення

Розподілені обчислення – це одна зі сфер комп'ютерних наук, що вивчає розподілені системи [9]. Розподілена система – це система, компоненти котрої знаходяться на різних комп'ютерах у мережі, комунікація і кооперація між якими відбувається шляхом обміну повідомленнями. Компоненти такої системи взаємодіють один з одним задля досягнення певної спільної мети.

До основних характеристик розподілених систем відносимо: одночасність роботи компонентів, відсутність глобальної синхронізації та незалежність відмови компонентів. Прикладами розподілених систем є системи з сервісно-орієнтованою архітектурою, масові багатокористувацькі онлайн-ігри, системи з одноранговою (peer-to-peer) архітектурою. Під розподіленими обчисленнями також розуміємо спосіб розв'язання трудомістких обчислювальних завдань із використанням багатьох комп'ютерів, об'єднаних у мережу.

Розподілені обчислення є окремим випадком паралельних обчислень, де одне обчислювальне завдання вирішується за допомогою декількох процесорів на одному або кількох комп'ютерах. Однією з вимог до завдання, яке вирішують за допомогою розподіленої системи, є можливість його розділення на підзавдання, що їх можуть обчислювати паралельно. Комп'ютерна програма, яку виконують на розподіленій системі, називається розподіленою. Існує багато реалізацій механізму комунікації між компонентами таких програм: HTTP, RPC тощо [9-12].

Розподілений алгоритм

Розподілений алгоритм – це алгоритм, створений для виконання на апаратному забезпеченні із взаємопов'язаними центральними процесорами. Розподілені алгоритми використовуються в різноманітних галузях розподіленого обчислення: телекомунікації, обчислювальних науках, розподіленій обробці даних, автоматизації виробництва.

Розподілені алгоритми – один із типів паралельних алгоритмів. Як правило, такі алгоритми виконуються одночасно, окремі частини алгоритму паралельно виконуються на незалежному обчислювальному апаратному та програмному забезпеченні з обмеженим знанням про діяльність інших частин алгоритму. Одна з найбільших проблем проєктування та реалізації розподілених алгоритмів полягає у втіленні вдалої координації поведінки незалежних частин алгоритму, включно з обробкою відмов незалежних вузлів і каналів їхньої комунікації.

Дизайн рішення

Наша робота подає високорівневий огляд розподіленої реалізації методу нейроеволюції наростаючої топології на основі реалізації SharpNEAT.

На рис. 3 зображено обчислювальні вузли під час роботи програми, а також

компоненти та об'єкти, які можуть бути виконані на цих вузлах. Для візуалізації використовуємо UML діаграму розгортання. Діаграма розгортання моделює фізичне розгортання артефактів на вузлах. Вузли зображені у вигляді прямокутних паралелепіпедів. Артефакти вузлів – як внутрішні прямокутники. Вузли можуть мати рівні вкладеності. Один вузол на такій діаграмі може представляти шаблон великої кількості фізичних вузлів, як-то кластер чи набір баз даних [13].

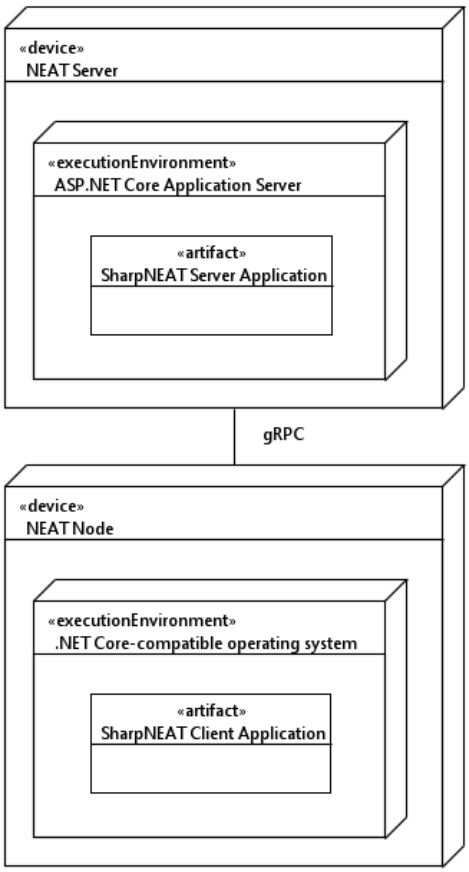

Рисунок 3. UML діаграма розгортання пропонованого рішення

Існує два типи вузлів: вузол-пристрій і вузол-середовище виконання. Вузли-пристрої – це фізичні обчислювальні ресурси, наприклад, комп'ютери та мобільні пристрої. Вузол середовища виконання – програмний обчислювальний ресурс, який виконують на батьківському вузлі, він надає сервіси й можливості для виконання іншого розгорнутого програмного забезпечення.

Наша реалізація включає два типи вузлів-пристроїв: NEAT Server та NEAT Client. NEAT Server – вузол-пристрій, що є головним сервером розподіленої реалізації методу нейроеволюції наростаючої топології. Цей сервер відповідає за виконання основної частини алгоритму NEAT, за управління та комунікацію з вузлами-клієнтами, а також за розподілення завдань. NEAT Client – вузол-пристрій, який є сукупністю пристроїв-клієнтів, відповідальних за частину оцінки придатності геномів в алгоритмі нейроеволюції наростаючої топології та за комунікацію з вузлом-сервером.

Наша реалізація включає два типи вузлів-середовищ виконання: ASP.NET Core Application Server та .NET Core-compatible operating system. Середовище виконання ASP.NET Core Application Server – .NET сумісна операційна система та реалізація ASP.NET Core серверу.

- ASP.NET Core – вільне та відкрите програмне забезпечення каркасу вебзастосунків, наступник ASP.NET від компанії Microsoft. ASP.NET Core є модульним фреймворком, що водночас підтримує виконання на операційній системі Windows з повноцінним фреймворком .NET Framework та на багатоплатформовій реалізації .NET [14].

- .NET Framework – програмний каркас, розроблений компанією Microsoft, що призначений переважно для виконання в середовищі сімейства операційних систем Microsoft Windows.

- .NET (попередньо .NET Core) – вільне та відкрите програмне забезпечення, програмний каркас для сімейств операційних систем Windows, Linux, macOS. Такий програмний каркас є багатоплатформовим наступником .NET Framework. Проєкт здебільшого підтримує та розробляє компанія Microsoft, випускають його під ліцензією MIT License.

Програмне забезпечення ASP.NET Core розгортається з такими вбудованими компонентами:

- Сервер Kestrel, багатоплатформна реалізація HTTP серверу. Kestrel надає найкращу доступну продуктивність виконання та використання пам'яті.

- Сервер IIS HTTP – сервер, що вбудований у головний процес для IIS.

IIS, Internet Information Services – вебсерверне програмне забезпечення, створене компанією Microsoft.

- Сервер HTTP.sys – ексклюзивний для сімейства операційних систем Windows HTTP сервер, що базується на драйвері ядра HTTP.sys та HTTP Server API.

На рис. 4 схематично зображено процес обробки мережевих запитів за участю компонентів Kestrel і коду цільового програмного забезпечення на базі ASP.NET Core:

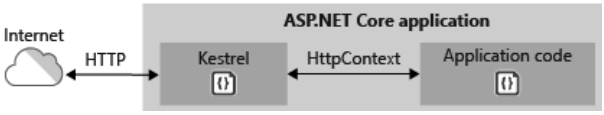

Рисунок 4. Схематичне зображення процесу обробки мережевих запитів

Ця реалізація включає два типи артефактів вузлів середовища виконання: SharpNEAT Server Application і SharpNEAT Client Application.

Артефакт SharpNEAT Server Application – це програмний застосунок на базі оригінального фреймворка SharpNEAT з модифікаціями й додатковими модулями для підтримки виконання в розподіленій системі. Ці модифікації та додаткові модулі відповідають за функції розподіленого виконання оцінки придатності геномів. Ця версія програмного застосунку виконує роль серверу – вона є центральним артефактом, відповідальним за виконання основної частини алгоритму NEAT, розподілення завдань між вузлами-клієнтами, комунікацію та управління ними.

Артефакт SharpNEAT Client Application також є програмним застосунком на базі оригінального фреймворка SharpNEAT із модифікаціями й додатковими модулями для підтримки виконання в розподіленій системі. Ця версія програмного застосунку виконує роль клієнта – вона є підпорядкованим артефактом-клієнтом, відповідальним за оцінку придатності отриманих завдань-геномів і комунікацію з вузлом-сервером.

На рис. 5 представлено UML діаграму послідовності взаємодії вказаних вузлів: NEAT Client і NEAT Server. Комунікація відбувається з використанням системи виклику віддалених процедур gRPC. Метод комунікації – потоковий, двосторонній. Порядок обробки повідомлень під час використання такого методу залежить від конкретної реалізації. Потоки незалежні: клієнт і сервер мають можливість надсилати та зчитувати повідомлення в будь-якому порядку. Наприклад, сервер може дочекатись отримання всіх повідомлень від клієнта перед надсиланням відповідей, або сервер і клієнт можуть почергово обмінюватися повідомленнями – сервер отримує запит, відповідає, отримує новий запит на основі відповіді й так далі.

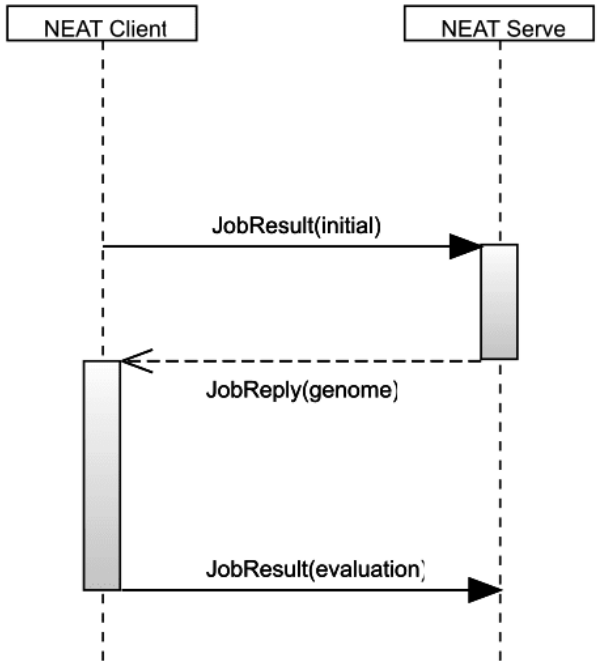

Рисунок 5. UML діаграма послідовності взаємодії вузлів

За такої реалізації існує два типи повідомлень: JobResult і JobReply. Комунікацію ініціює вузол-клієнт шляхом виклику RPC методу AcquireJob. Сервер отримує метадані клієнта та інформацію про метод. Параметри початкового повідомлення JobResult відсутні, тобто не містять ніяких результатів попередньої оцінки придатності. Сервер відповідає пакетом завдань-геномів JobReply за першої можливості. Після отримання завдання клієнт виконує оцінку придатності геномів і знову надсилає повідомлення JobResult, проте з даними оцінки

придатності. Після цього цикл JobReply (genome) – JobResult (evaluation) нескінченно повторюється.

Високорівневий огляд реалізації алгоритму нейроеволюції наростаючої топології у програмному фреймворку SharpNEAT був представлений на рис. 1. На рис. 6 зображено високорівневий огляд реалізації процедури оцінки чергового покоління. Саме процедура оцінювання виплоду є завданням розподілених обчислень нашої роботи.

Рисунок 6. Високорівневий огляд реалізації процедури оцінки чергового покоління

Виклик віддалених процедур

У сфері розподілених обчислень виклик віддалених процедур – це протокол, який дозволяє програмному забезпеченню викликати процедури (підпрограми) в іншому адресному просторі. Зазвичай під іншим адресним простором розуміють інший комп'ютер у спільній мережі. Однією з особливостей протоколу є відсутність необхідності розробки деталей комунікації. Більше того, такий протокол дозволяє реалізовувати міжпроцесорну комунікацію для стеків спочатку несумісних технологій програмного забезпечення. Виклик віддалених процедур – одна з форм клієнт-серверної взаємодії.

gRPC (Google Remote Procedure Calls) – вільна й відкрита програмна система виклику віддалених процедур. Система gRPC уперше була розроблена компанією Google. Вона використовує протокол HTTP/2 як транспорт, а Protocol Buffers (Protobuf) – як мову опису інтерфейсів. gRPC надає такі функції: аутентифікація, двостороння потокова комунікація, управління потоком передачі даних тощо. Існує багато реалізацій gRPC, які генерують сполучний код для легкої інтеграції в програмне забезпечення на цільових платформах і мовах [15].

Protobuf

Protocol Buffers – вільна й відкрита багатоплатформова бібліотека серіалізації структурованих даних. Бібліотека Protocol Buffers розроблена компанією Google. Структури даних у цій бібліотеці називають повідомленнями. Повідомлення і зв'язані сервіси описані в proto-файлі. Повідомлення серіалізуються в компактний, зворотно сумісний, бінарний, але такий, що не описує себе, формат [16].

Система gRPC використовує формат Protocol Buffers для серіалізації даних. На відміну від класичних прикладних програмних інтерфейсів HTTP JSON API, Protocol Buffers мають чітку єдину специфікацію, що дозволяє системі gRPC узгоджено працювати з даними між різними платформами та реалізаціями. Формат Protocol Buffers також пропонує високі показники продуктивності в питаннях серіалізації, десеріалізації і транспорту внаслідок бінарної природи формату повідомлень.

На наступному фрагменті зображено приклад Protocol Buffers специфікації сервісу й набору повідомлень розподіленої реалізації методу нейроеволюції наростаючої топології нашої роботи.

```
    service NeatProtoService {
       rpc AcquireJob (stream
JobResult)   returns   (stream
JobReply);
    }

    message JobResult {
      repeated GenomeTaskResult
genomeTaskResults = 1;
    }

    message JobReply {
         repeated  GenomeTask
genomeTasks = 1;
      string assemblyName = 2;
      string typeName = 3;
    }
```

```
message GenomeTask {
  string id = 1;
  repeated bytes genomes = 2;
}

message GenomeTaskResult {
  string id = 1;
  repeated double fitness = 2;
}
```

Схема має таку специфікацію сервісів та повідомлень:

- Повідомлення JobResult містить набір даних про виконану вузлом-клієнтом оцінку придатності.

- Повідомлення JobReply включає дані про завдання для клієнта, дані для конфігурації експерименту й алгоритму нейроеволюції наростаючої топології на клієнтській стороні та набір завдань GenomeTask.

- Повідомлення GenomeTask містить унікальний серверний ідентифікатор завдання і пакет серіалізованих геномів для оцінки придатності.

- Повідомлення GenomeTaskResult включає дані оцінки пакету геномів та унікальний серверний ідентифікатор завдання.

- Сервіс NeatProtoService містить специфікацію виклику віддалених процедур, а саме потоковий двосторонній метод AcquireJob, що визначає порядок використання повідомлень JobResult і JobReply.

Деталі реалізації

На рис. 7 зображено високорівневу діаграму модулів програмного застосунку серверної частини рішення.

- Модуль SharpNeat.GridServer відповідає за програмний сервер розподіленої реалізації методу нейроеволюції наростаючої топології. Цей модуль оркеструє виконання основної частини алгоритму NEAT, здійснює управління та комунікацію з вузлами-клієнтами, а також розподіляє завдання.

- Модуль SharpNeat.GridCommon містить спільні для програмних модулів SharpNeat.GridServer і SharpNeat.GridClient файли конфігурації головного завдання, порядок їхньої обробки під час складання проєкту, а також спільний proto-файл.

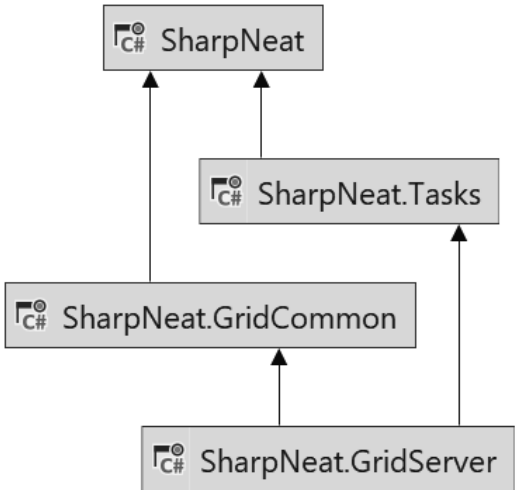

Рисунок 7. Високорівнева діаграма модулів програмного застосунку серверної частини рішення

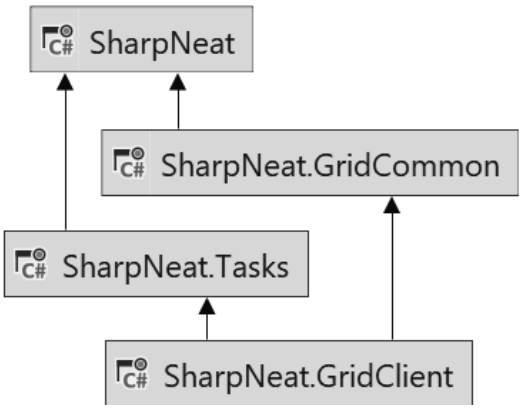

Рисунок 8. Високорівнева діаграма модулів програмного застосунку клієнтської частини рішення

На рис. 8 зображено високорівневу діаграму модулів програмного застосунку клієнтської частини рішення.

На рис. 9 зображено загальну високорівневу діаграму модулів програмного застосунку всього рішення.

Для забезпечення максимальної продуктивності розподілення завдань наша реалізація використовує такі особливості:

- патерн async/await;

- пакет System.Threading;

- пакет System.Threading.Tasks (Parallel, Task);

- пакет System.Collections.Concurrent (BlockingCollection, ConcurrentDictionary);
- пакет System.Linq.

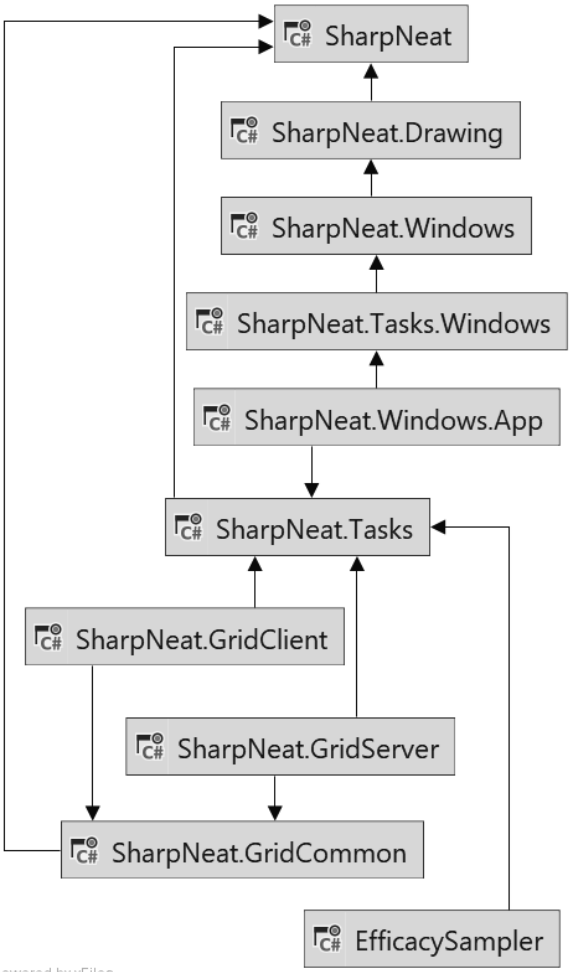

Рисунок 9. Загальна високорівнева діаграма модулів програмного застосунку рішення

Робота з вузлами й завданнями відбувається паралельно. Реалізація серверної частини є багатонитково-безпечною, що дозволяє уникати побічних негативних наслідків і непередбачуваної поведінки.

Бенчмаркінг

Як експеримент для бенчмаркінгу було реалізовано завдання двійкового мультиплексора на 20 входів (мультиплексор 16 в 1) [17]. На рис. 10 зображено умовну схему такого мультиплексора. Символом A позначені входи даних, символом B – адресні входи, Y – вихід. Нейронна мережа в цій задачі має 20 входів (20 нейронів вхідного шару), 4 з яких – адресні, а 16 – входи для даних. Усі входи приймають двійкові значення (0 або 1). У нейронної мережі є один вихід (один нейрон вихідного шару). Двійкова адреса подається на 4 адресні входи, що репрезентує вибір одного з 16 значень входів для даних. Оцінка складається з вичерпної перевірки нейронної мережі на кожній з 1048576 (2^{20}) можливих комбінацій входів. Вихідне значення нейронної мережі повинне збігатися зі значенням одного зі входів даних, що представлений двійковою адресою з адресних входів. Вихідне значення менше, ніж 0,5 уважають двійковим нулем, вихідне значення більше або рівне 0,5 – двійковою одиницею. Значення оцінки (придатності) адитивно розраховують у результаті ретельної перевірки. У разі відсутності помилок присуджують додаткову винагороду. Це завдання обране з огляду на одночасну простоту реалізації і складність виконання, що допоможе продемонструвати розподілення важких завдань між багатьма вузлами та їхню ефективну оркестрацію.

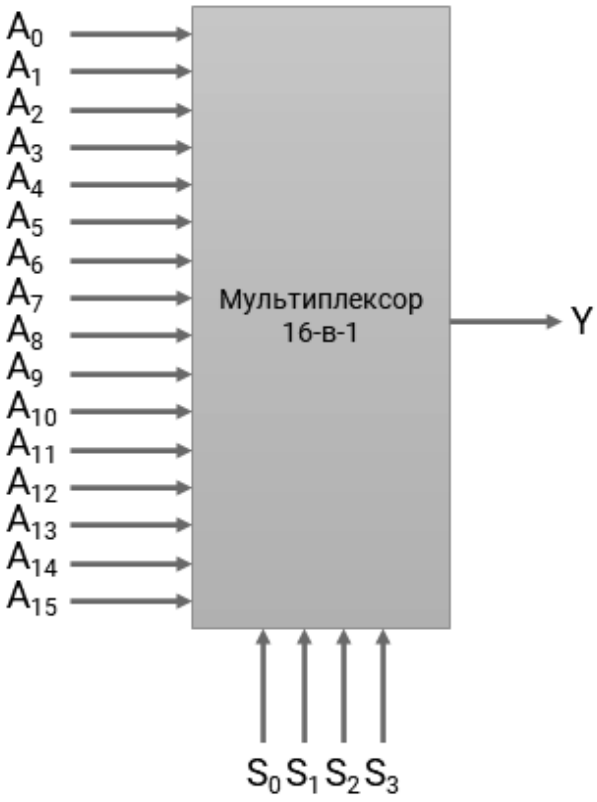

Рисунок 10. Умовна схема мультиплексора типу 16-в-1

Платформа хмарних обчислень Amazon Web Services пропонує широкий набір сервісів, можливостей і функцій для розгортання .NET додатків у хмарі AWS. Прикладами таких сервісів є контейнери Amazon Elastic Container Service (ECS) та Amazon Elastic Kubernetes Service (EKS), AWS Fargate, AWS Lambda, Amazon EC2 тощо [18].

Сімейства віртуальних машин на базі процесорів AWS Graviton2 пропонують вагомий виграш у продуктивності .NET додатків порівняно з еквівалентними віртуальними машинами сімейства Intel x86. Окрім цього, фреймворк .NET версії 5 включає ряд оптимізацій для архітектури ARM64.

Як обчислювальні вузли для бенчмаркінгу цього рішення використовувались віртуальні машини сервісу хмарних обчислювальних потужностей Amazon Elastic Compute Cloud (Amazon EC2) c6g.medium. Віртуальні машини Amazon EC2 C6g працюють на базі процесорів AWS Graviton2 з архітектурою ARM64. Віртуальні машини класу c6g.medium пропонують 1 віртуальний центральний процесор, 2 гігабайти оперативної пам'яті, до 10 Гбіт/с мережевої пропускної здатності та до 4750 Мбіт/с пропускної здатності сховища. Віртуальні машини такого типу зазвичай використовують для високопродуктивних обчислень, пакетної обробки даних, кодування відео, наукового моделювання, розподіленої аналітики, машинного навчання на центральних процесорах. Як вузол-сервер також може використовуватися одна віртуальна машина c6g.medium.

Задля створення умов середовища бенчмаркінгу, максимально наближених до реальних, вузли-клієнти й вузол-сервер були географічно віддалені на відстань приблизно в 500 кілометрів. Географічне розділення було досягнуто завдяки використанню AWS Availability Zone. Ця особливість допоможе відтворити реальну поведінку мережевої взаємодії вузлів, на відміну від використання високошвидкісних локальних мереж.

Ми використали таку конфігурацію віртуальних машин:

- операційна система: Ubuntu Server 20.04 LTS (HVM);
- накопичувач: EBS General Purpose (SSD);
- сімейство: c6g;
- кількість віртуальних центральних процесорів: 1;
- обсяг оперативної пам'яті: 2 Гб.

Для пакетного розгортання інфраструктури були використані функції Launch templates та AMIs. На рис. 11 зображено діаграму архітектури хмарного розгортання рішення на AWS.

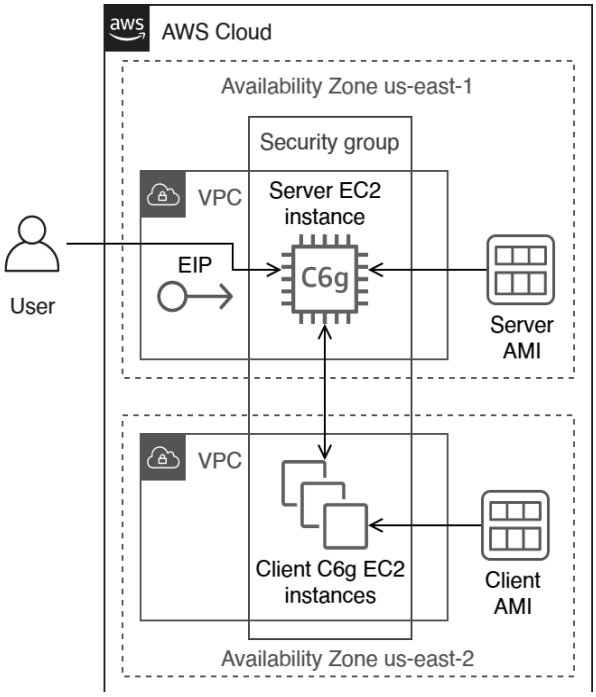

Рисунок 11. Діаграма архітектури хмарного розгортання рішення на AWS

Основні елементи інфраструктури:

- Availability Zone – ізольовані локації в різних географічних регіонах.
- Security group – віртуальний міжмережевий екран для контролю вхідного й вихідного трафіку.
- VPC (Amazon Virtual Private Cloud, Amazon VPC) – масштабована віртуальна приватна хмара, пул спільно використовуваних обчислювальних ресурсів.
- EIP (Elastic IP address) – статична, публічна IPv4 адреса, асоційована з віртуальною машиною.
- AMI (Amazon Machine Image) – підтримуваний образ операційної системи для використання в Amazon Elastic

Compute Cloud (Amazon EC2). AMI надає інформацію для запуску віртуальної машини.

- EC2 (Amazon Elastic Compute Cloud, Amazon EC2) – масштабовані обчислювальні ресурси в хмарі AWS.

З метою оптимізації продуктивності рішення, рівномірного розподілу завдань між вузлами та оптимального використання обчислювальних ресурсів була реалізована підтримка пакетної оцінки геномів. Тож, з'являється можливість зменшення накладних витрат обчислювальних ресурсів вузлів-клієнтів і вузла-сервера на частий обмін одиничними завданнями та їхніми результатами. Кількість завдань в одному пакеті була розрахована за такою формулою:

де: $size_{batch} = \frac{size_{population}}{size_{fleet}}$,

- $size_{batch}$ – кількість завдань в одному пакеті;
- $size_{population}$ – максимальна кількість завдань на оцінку одного покоління;
- $size_{fleet}$ – кількість обчислювальних вузлів-клієнтів.

Кількість завдань в одному пакеті в представленому рішенні конфігурується перед запуском серверного програмного додатка.

На рис. 12 зображено графік залежності швидкості оцінки від середньої кількості завдань у пакеті. Як видно з графіку, для подібного програмного рішення, середовища й завдання накладні витрати на взаємодію між вузлами нехтуються вже на найменших розмірах пакетів і дозволяють масштабувати розмір пакету з очікуваною продуктивністю. Для такого бенчмаркінгу використовувалися пакети розміром від 6 до 3000 завдань.

На рис. 13 зображено об'єднаний графік залежності середнього розміру пакету завдань і загальної швидкості оцінювання від розміру флоту вузлів-клієнтів.

На рис. 14 зображено графік залежності загальної швидкості оцінювання від розміру флоту вузлів-клієнтів.

Подані графіки засвідчують, що, використовуючи запропоноване розподілене рішення, швидкість виконання методу нейроеволюції наростаючої топології в частині оцінювання згенерованих нейронних мереж на прикладі розглянутого завдання зросла з 13 оцінювань за секунду до майже 3790 оцінювань за секунду при використанні 1024 хмарних обчислювальних вузлів зазначеної конфігурації.

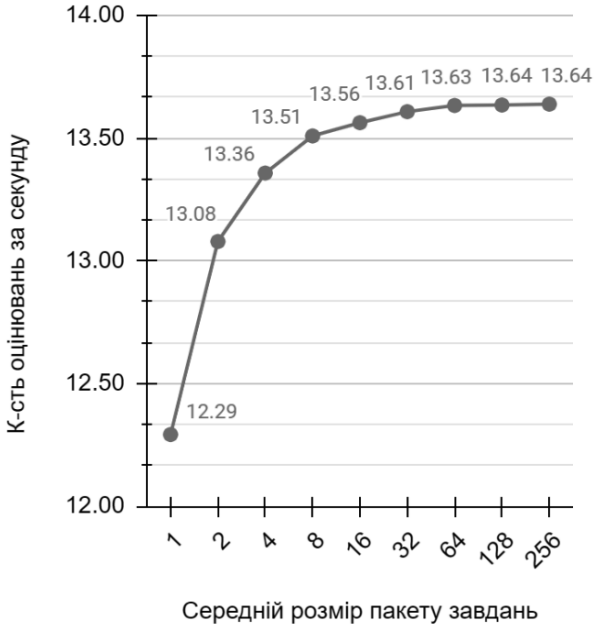

Рисунок 12. Графік залежності швидкості оцінки від середньої кількості завдань в пакеті

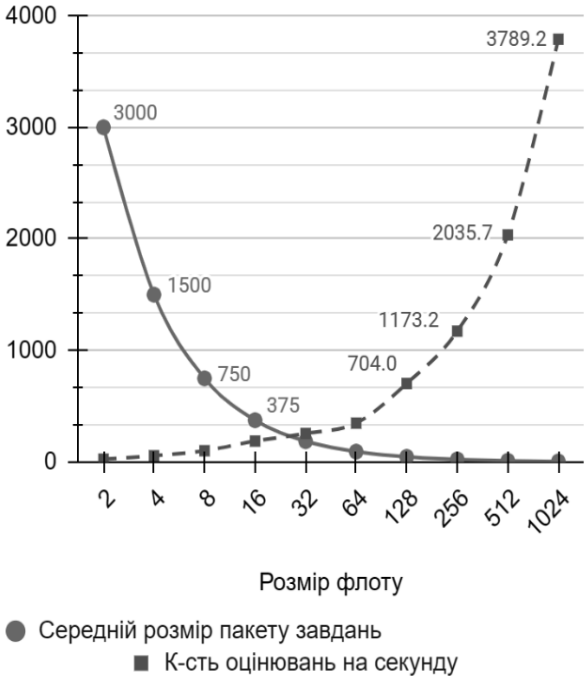

Рисунок 13. Об'єднаний графік залежності середнього розміру пакету завдань і загальної швидкості оцінювання від розміру флоту вузлів-клієнтів

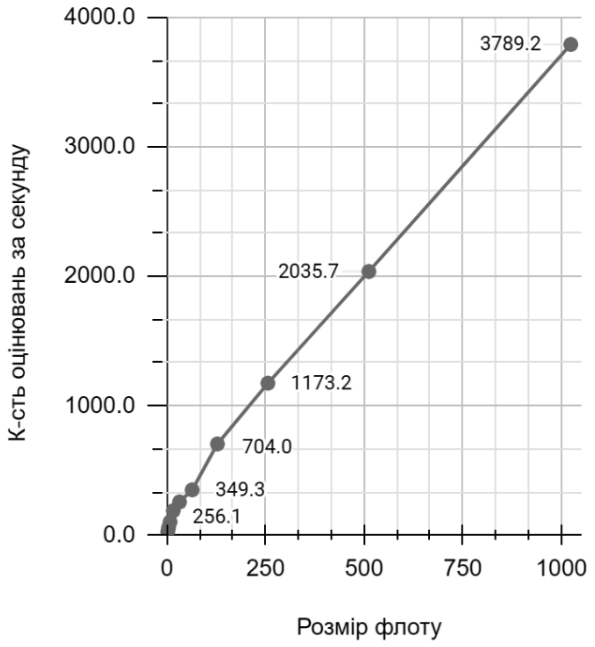

Рисунок 14. Графік залежності загальної швидкості оцінювання від розміру флоту вузлів-клієнтів

На рис. 15 зображено графік залежності швидкості оцінювання від порядкового номера покоління.

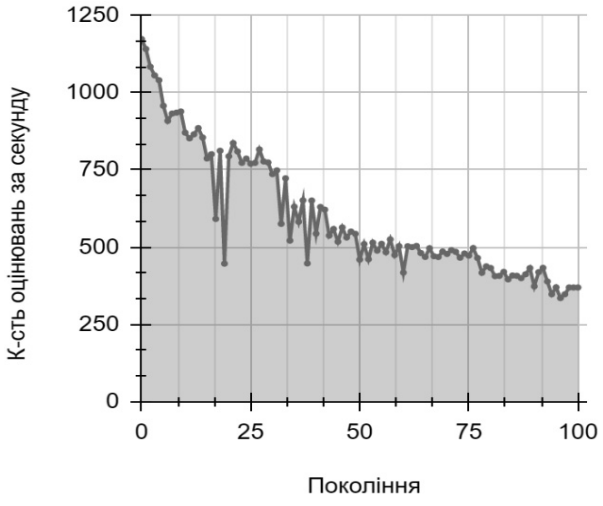

Рисунок 15. Графік залежності швидкості оцінювання від порядкового номера покоління.

Як видно з графіку, швидкість оцінювання зменшується з ходом еволюції. Це пов'язано з тим, що в процесі еволюції геноми (а, як наслідок, і представлені нейронні мережі) набувають об'ємніших і складніших топологій, тобто збільшується кількість шарів, нейронів і зв'язків між ними.

Висновки

У роботі запропоновано нову розподілену реалізацію методу нейроеволюції наростаючої топології, що за наявності достатніх обчислювальних ресурсів дозволяє радикально збільшити швидкість знаходження оптимальних конфігурацій нейронних мереж і допомагає обійти проблему повільної конвергенції до оптимальних результатів.

Література

1. Evolution 101: Neuroevolution | BEACON. BEACON | An NSF Center for the Study of Evolution in Action. URL: https://beacon-center.org/blog/2012/08/13/evolution-101-neuroevolution/ (дата звернення: 08.08.2021).
2. Субботін С. О., Олійник А. О., Олійник О. О. Неітеративні, еволюційні та мультиагентні методи синтезу нечіткологічних і нейромережних моделей : монографія / ред. Субботін С. О. Запоріжжя : ЗНТУ, 2009. 375 с.
3. Stanley K. O. Efficient evolution of neural networks through complexification : Thesis. 2004. URL: http://hdl.handle.net/2152/1266 (дата звернення: 08.08.2021).
4. NeuroEvolution of Augmenting Topologies. Department of Computer Science, College of Engineering and Computer Science@UCF. URL: http://www.cs.ucf.edu/~kstanley/neat.html (дата звернення: 08.08.2021).
5. Stanley K. O., Miikkulainen R. Evolving Neural Networks through Augmenting Topologies. Evolutionary Computation. 2002. Т. 10, № 2. С. 99–127. URL: https://doi.org/10.1162/106365602320169811 (дата звернення: 08.08.2021).
6. Stanley K. O., Bryant B. D., Miikkulainen R. Real-Time Neuroevolution in the NERO Video Game. IEEE Transactions on Evolutionary Computation. 2005. Т. 9, № 6. С. 653–668. URL: https://doi.org/10.1109/tevc.2005.856210 (дата звернення: 08.08.2021).
7. Stanley K. O., Miikkulainen R. Competitive Coevolution through Evolutionary Complexification. Journal of Artificial Intel-

ligence Research. 2004. T. 21. C. 63–100. URL: https://doi.org/10.1613/jair.1338 (дата звернення: 08.08.2021).
8. Green C. SharpNEAT Neuroevolution Framework. SharpNEAT Neuroevolution Framework. URL: https://sharpneat.sourceforge.io/ (дата звернення: 08.08.2021).
9. Andrews G. R. Foundations of multithreaded, parallel, and distributed programming. Reading, Mass : Addison-Wesley, 2000. 664 с.
10. Arora S. Computational complexity: A modern approach. Cambridge : Cambridge University Press, 2009.
11. Lynch N. A. Distributed algorithms. San Francisco, Calif : Morgan Kaufmann, 1997. 872 с.
12. Peleg D. Distributed computing: A locality-sensitive approach. Philadelphia : Society for Industrial and Applied Mathematics, 2000.
13. Booch G., Rumbaugh J., Jacobson I. Unified Modeling Language User Guide, The (2nd Edition) (The Addison-Wesley Object Technology Series). 2-ге вид. Addison-Wesley Professional, 2005. 496 с.
14. ASP.NET documentation. Developer tools, technical documentation and coding examples | Microsoft Docs. URL: https://docs.microsoft.com/en-us/aspnet/core/?view=aspnetcore-5.0 (дата звернення: 08.08.2021).
15. Introduction to gRPC. gRPC. URL: https://grpc.io/docs/what-is-grpc/introduction/ (дата звернення: 08.08.2021).
16. Language Guide | Protocol Buffers | Google Developers. Google Developers. URL: https://developers.google.com/protocol-buffers/docs/overview (дата звернення: 08.08.2021).
17. The 11-multiplexer Problem. GEP: Home. URL: https://www.gene-expression-programming.com/webpapers/Ferreira-CS2001/Section6/SS5/SSS2.htm (дата звернення: 08.08.2021).
18. Powering .NET 5 with AWS Graviton2: Benchmarks | Amazon Web Services. Amazon Web Services. URL: https://aws.amazon.com/ru/blogs/compute/powering-net-5-with-aws-graviton2-benchmark-results/ (дата звернення: 08.08.2021).

References

1. Evolution 101: Neuroevolution | BEACON. BEACON | An NSF Center for the Study of Evolution in Action. URL: https://beacon-center.org/blog/2012/08/13/evolution-101-neuroevolution/ (date of access: 08.08.2021).
2. Subbotin S., Oliinyk A., Oliinyk O. Nonitterative, Evolutionary, and Multiagent Methods of Synthesis of Fuzzy Logic and Neural Network Models / ed. by S. O. Subbotin. Zaporizhzhya : ZNTU, 2009. 375 p.
3. Stanley K. O. Efficient evolution of neural networks through complexification : Thesis. 2004. URL: http://hdl.handle.net/2152/1266 (date of access: 08.08.2021).
4. NeuroEvolution of Augmenting Topologies. Department of Computer Science, College of Engineering and Computer Science@UCF. URL: http://www.cs.ucf.edu/~kstanley/neat.html (date of access: 08.08.2021).
5. Stanley K. O., Miikkulainen R. Evolving Neural Networks through Augmenting Topologies. Evolutionary Computation. 2002. Vol. 10, no. 2. P. 99–127. URL: https://doi.org/10.1162/106365602320169811 (date of access: 08.08.2021).
6. Stanley K. O., Bryant B. D., Miikkulainen R. Real-Time Neuroevolution in the NERO Video Game. IEEE Transactions on Evolutionary Computation. 2005. Vol. 9, no. 6. P. 653–668. URL: https://doi.org/10.1109/tevc.2005.856210 (date of access: 08.08.2021).
7. Stanley K. O., Miikkulainen R. Competitive Coevolution through Evolutionary Complexification. Journal of Artificial Intelligence Research. 2004. Vol. 21. P. 63–100. URL: https://doi.org/10.1613/jair.1338 (date of access: 08.08.2021).
8. Green C. SharpNEAT Neuroevolution Framework. SharpNEAT Neuroevolution Framework. URL: https://sharpneat.sourceforge.io/ (date of access: 08.08.2021).
9. Andrews G. R. Foundations of multithreaded, parallel, and distributed programming. Reading, Mass : Addison-Wesley, 2000. 664 p.
10. Arora S. Computational complexity: A modern approach. Cambridge : Cambridge University Press, 2009.
11. Lynch N. A. Distributed algorithms. San Francisco, Calif : Morgan Kaufmann, 1997. 872 p.

12. Peleg D. Distributed computing: A locality-sensitive approach. Philadelphia : Society for Industrial and Applied Mathematics, 2000.
13. Booch G., Rumbaugh J., Jacobson I. Unified Modeling Language User Guide, The (2nd Edition) (The Addison-Wesley Object Technology Series). 2nd ed. Addison-Wesley Professional, 2005. 496 p.
14. ASP.NET documentation. Developer tools, technical documentation and coding examples | Microsoft Docs. URL: https://docs.microsoft.com/en-us/aspnet/core/?view=aspnetcore-5.0 (date of access: 08.08.2021).
15. Introduction to gRPC. gRPC. URL: https://grpc.io/docs/what-is-grpc/introduction/ (date of access: 08.08.2021).
16. Language Guide | Protocol Buffers | Google Developers. Google Developers. URL: https://developers.google.com/protocol-buffers/docs/overview (date of access: 08.08.2021).
17. The 11-multiplexer Problem. GEP: Home. URL: https://www.gene-expression-programming.com/webpapers/Ferreira-CS2001/Section6/SS5/SSS2.htm (date of access: 08.08.2021).
18. Powering .NET 5 with AWS Graviton2: Benchmarks | Amazon Web Services. Amazon Web Services. URL: https://aws.amazon.com/ru/blogs/compute/powering-net-5-with-aws-graviton2-benchmark-results/ (date of access: 08.08.2021).

Одержано 11.08.2021

***Про авторів*:**

Ашур Ілля Зін-Еддінович,
аспірант 3 курсу в
НТУУ “КПІ імені Ігоря Сікорського”,
https://orcid.org/0000-0003-2348-8777

Дорошенко Анатолій Юхимович,
доктор фізико-математичних наук,
професор, завідувач відділу теорії
комп’ютерних обчислень,
професор кафедри інформаційних
систем та технологій НТУУ
“КПІ імені Ігоря Сікорського”.
Кількість наукових публікацій в
українських виданнях – понад 190.
Кількість наукових публікацій в
зарубіжних виданнях – понад 80.
Індекс Хірша – 6.
http://orcid.org/0000-0002-8435-1451

Місце роботи авторів:

Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського»,
проспект Перемоги 37
та Інститут програмних систем
НАН України,
03187, м. Київ-187,
проспект Академіка Глушкова, 40.
Тел.: (044) 526 3559
E-mail: ilyaachour@gmail.com, doroshenkoanatoliy2@gmail.com