ВІСНИК ПРИАЗОВСЬКОГО ДЕРЖАВНОГО ТЕХНІЧНОГО УНІВЕРСИТЕТУ
2023р. Серія: Технічні науки Вип. 47
p-ISSN: 2225-6733; e-ISSN: 2519-271X

122 КОМП'ЮТЕРНІ НАУКИ ТА ІНФОРМАЦІЙНІ ТЕХНОЛОГІЇ

УДК 004.75 **doi: 10.31498/2225-6733.47.2023.299923**

© Балалаєва О.Ю.[1], Марченко І.Ф.[2], Коротенко Г.М.[3], Бешта Д.О.[4], Пікуз А.К.[5]

ДОСЛІДЖЕННЯ ЕФЕКТИВНОСТІ РОБОТИ СЕРІАЛІЗАТОРІВ ДАНИХ МОВИ ПРОГРАМУВАННЯ C# ЗА ДОПОМОГОЮ РОЗРОБЛЕНОГО ПРОГРАМНОГО ПРОДУКТУ ДЛЯ ТЕСТУВАННЯ

У статті розглянуто питання використання серіалізаторів даних для реалізації проєктів, пов'язаних з обробкою великих обсягів даних, а також із підтримкою високошвидкісної передачі даних у розподілених системах. Показано, що в даному контексті вибір найбільш ефективного механізму серіалізації має критичне значення для забезпечення продуктивності та масштабованості додатків. Метою даної роботи є дослідження ефективності роботи серіалізаторів даних мови програмування C# за допомогою розробки програмного продукту для тестування серіалізаторів з використанням різних за обсягом і типом даних об'єктів. Проведено огляд наукових досліджень застосування різних форматів серіалізації даних: XML, JSON, BSON, MessagePack, Smile, Protocol Buffers, Flat Buffers, Apache Thrift. Зроблено висновок, що найбільш популярними на сьогодні є формати XML та JSON, виконано їх порівняльний аналіз. Обґрунтовано доцільність використання JSON-формату серіалізації, що обумовлено його безпечністю у порівнянні з бінарним форматом, меншим розміром у порівнянні з XML-форматом, а також підтримкою більшістю засобів розробки програмних продуктів. Обрано платформу .NET, яка надає стандартні інструменти для JSON-серіалізації мови програмування C#, а саме: System.Runtime.Serialize.Json та System.Text.Json, які постачаються за замовчанням. Проаналізовано найбільш популярні програмні рішень для серіалізації об'єктів C#, показано доцільність тестування таких серіалізаторів, як Jil, Json.NET, Utf8Json, SpanJson та стандартних серіалізаторів з метою виявлення переваг та недоліків їхнього використання для реалізації конкретних задач та проєктів. Для створення програми-тестувальника обрано бібліотеку мови програмування C# BenchmarkDotNet. Зазначено, що даний фреймворк платформи .NET дозволяє перетворювати методи на тести та створювати тестування ефективності завдяки потужному статистичному механізму. Наведено діаграму класів та діаграму компонентів розробленого програмного забезпечення. Проведено дослідження 5 серіалізаторів даних, які включали виконання 7 експериментів з серіалізації об'єктів з різними типами даних. Проаналізовано витрати часу та оперативної пам'яті при серіалізації малого та великого об'єктів; об'єктів, що містять одновимірні, двовимірні та тривимірні масиви натуральних чисел, об'єкта зі складним ланцюгом спадкування класів, а також об'єкта, що містить словник. Результати

[1] *канд. техн. наук, доцент, ДВНЗ «Приазовський державний технічний університет», м. Дніпро/Маріуполь, ORCID: 0000-0003-1461-4399, balalaevaeu@gmail.com*
[2] *канд. техн. наук, доцент, ДВНЗ «Приазовський державний технічний університет», м. Дніпро/Маріуполь, ORCID: 0000-0002-4566-3866, irsa665@gmail.com*
[3] *д-р техн. наук, професор, Національний технічний університет «Дніпровська політехніка», м. Дніпро, ORCID: 0000-0003-3774-5260, korotenko.g.m@nmu.one*
[4] *канд. техн. наук, Національний технічний університет «Дніпровська політехніка», м. Дніпро, ORCID: 0000-0003-2848-2737, beshta.d.o@nmu.one*
[5] *магістрант, ДВНЗ «Приазовський державний технічний університет», м. Дніпро, lokrastr@gmail.com*

p-ISSN: 2225-6733; e-ISSN: 2519-271X

експериментальних досліджень показали залежність ефективності серіалізаторів від типу та обсягу даних, які потрібно серіалізувати. Зроблено висновок, що не існує універсального серіалізатора, який буде показувати найкращі результати в усіх випадках. Надано рекомендації щодо використання різних серіалізаторів з урахуванням вимог конкретного проєкту.
Ключові слова: *серіалізація даних, серіалізатор, мова програмування C#, формат JSON, платформа .NET.*

O. Balalaieva, I. Marchenko, G. Korotenko, D. Beshta, A. Pikuz. Performance research of C# programming language data serializers using the developed software product for testing. *The article deals with the issue of using data serializers for the implementation of projects related to the processing of large volumes of data, as well as the support of high-speed data transmission in distributed systems. It is shown that in this context, the choice of the most effective serialization mechanism is critical for ensuring the performance and scalability of applications. The purpose of this work is to study the effectiveness of data serializers of the C# programming language by developing a software product for testing serializers using objects of different size and type. A review of scientific research on the use of various data serialization formats: XML, JSON, BSON, MessagePack, Smile, Protocol Buffers, Flat Buffers, Apache Thrift was conducted. It was concluded that XML and JSON formats are the most popular today, and their comparative analysis was performed. The expediency of using the JSON serialization format is substantiated, which is due to its safety compared to the binary format, its smaller size compared to the XML format, as well as the support of most software development tools. The .NET framework is chosen, which provides standard tools for JSON serialization of the C# programming language, namely: System.Runtime.Serialize.Json and System.Text.Json, which are supplied by default. The most popular software solutions for serializing C# objects are analyzed, the feasibility of testing such serializers as Jil, Json.NET, Utf8Json, SpanJson and standard serializers is shown in order to identify the advantages and disadvantages of their use for the implementation of specific tasks and projects. The C# BenchmarkDotNet programming language library was chosen to create the tester program. It is noted that this framework of the .NET platform allows you to convert methods into tests and create performance testing thanks to a powerful statistical mechanism. A class diagram and a component diagram of the developed software are given. A study of 5 data serializers was conducted, which included the execution of 7 experiments on serialization of objects with different types of data. The consumption of time and working memory during serialization of small and large objects was analyzed; objects containing one-dimensional, two-dimensional and three-dimensional arrays of natural numbers, an object with a complex chain of class inheritance, as well as an object containing a dictionary. The results of experimental studies showed the dependence of the effectiveness of serializers on the type and volume of data to be serialized. It is concluded that there is no one-size-fits-all serializer that will perform best in all cases. Recommendations for the use of various serializers are provided, taking into account the requirements of a specific project.*
Key words: *data serialization, serializer, C# programming language, JSON format, .NET platform.*

Постановка проблеми. При розробці програмних продуктів у сучасному світі інформаційних технологій центральну роль грають дані. Серіалізація є необхідним, надійним і зручним у застосуванні механізмом для зберігання, передачі та обміну даними між додатками, є невід'ємною частиною розробки додатків та може істотно впливати на їх продуктивність.

Серіалізація – це процес переведення будь-якої структури даних в послідовність бітів. Зворотною до операції серіалізації є операція десеріалізації (структуризації) – відновлення початкового стану структури даних з бітової послідовності. Ця послідовність може бути як бінарним представленням цих даних, так і текстовим [1]. Використання цієї технології дозволяє зручно зберігати данні великого обсягу у бажаному з доступних форматів. Максимальний термін існування будь-якого об'єкта при звичайному виконанні програми обмежений терміном виконання

p-ISSN: 2225-6733; e-ISSN: 2519-271X

цієї програми – від запуску до завершення її роботи. Серіалізація дає можливості для збереження стану екземпляра об'єкта поза циклом виконання програми, у межах якої цей екземпляр об'єкта створюється. Результатом роботи процедури серіалізації є форма, яку можна зберігати або передати між різними програмами, що виконуються як на одному пристрої, так і між програмними продуктами, що працюють на різних пристроях. При цьому реалізується ідея кросплатформності – немає значення, на якій операційній системі працює програмний продукт, серіалізація перетворює потрібний об'єкт або дерево об'єктів на потік байтів, який може бути відновлений на будь-якій операційній системі. Серіалізація необхідна для перетворення об'єкта на текстовий формат.

При роботі з великими обсягами даних або з високонавантаженими системами вибір механізму серіалізації є критично важливим, причому ефективність її застосування буде залежати від багатьох факторів, таких як швидкість серіалізації та десеріалізації, обсяг серіалізованих даних та споживання ресурсів тощо. Вибір того чи іншого серіалізатора залежить не тільки від технологій, на яких він базується, але й від даних, які безпосередньо потрібно серіалізувати у конкретному проєкті, а також наявних можливостей технічної бази. Саме тому використання програми-тестувальника дозволить визначити ефективний серіалізатор для кожного конкретного випадку використання даних.

Метою даної роботи є дослідження ефективності роботи серіалізаторів даних мови програмування C# за допомогою розробки програмного продукту для тестування серіалізаторів з використанням різних за обсягом і типом даних об’єктів..

Аналіз останніх досліджень і публікацій. Вирішити проблему вибору протоколу серіалізації для розробки довільного програмного забезпечення можна лише після аналізу основних переваг та недоліків існуючих протоколів та визначення рекомендованих сфер їх застосування, що дозволить обґрунтовано підійти до вибору протоколу при реалізації конкретного проєкту.

Автори [1] порівнюють роботу таких найбільш популярних протоколів серіалізації даних, як JSON і XML, з Protobus, при цьому в якості основних недоліків перших двох зазначають наявність лише низькорівневих інструментів для роботи з ними, завеликі обсяги серіалізованих даних та відсутність гарантії цілісності та захищеності даних. У роботі також виокремлено й переваги усіх трьох серіалізаторів. Наприклад, XML дозволяє забезпечити максимальну сумісність з існуючими системами, а JSON та Protobus задовольняють вимоги при серіалізації складних типів об’єктів в автоматизованих системах з підвищеними вимогами до продуктивності, хоча зазначається, що Protobuf має перевагу з точки зору безпеки передачі даних.

У статті [2] проведено порівняльний аналіз показників роботи більш широкого переліку текстових (JSON, XML) та бінарних форматів серіалізації даних (з серіалізацією схеми даних – BSON, Smile, MessagePack, без серіалізації схеми даних – ThriftBin, Compact, Protobus, Flatbuf), при цьому увагу зосереджено на доцільності використання додаткової компресії при серіалізації/десеріалізації даних. Авторами зроблено висновок, що такі бінарні формати серіалізації даних, як BSON та MessagePac, потребують більше часу на серіалізацію/десеріалізацію, ніж такий текстовий формат, як JSON. Авторами також зазначається, що при застосуванні компресії розмір повідомлень, серіалізованих у вищенаведених форматах, майже не відрізняються.

У роботі [3] проведено аналіз тих же серіалізаторів, при цьому зроблено висновок, що хоча й для кодування малих повідомлень найбільш ефективним форматом є Protocol Buffers, але це супроводжується суттєвим збільшенням розміру скомпільованої програми. Авторами зазначається, що застосування бінарних протоколів зі строгою схемою є більш складним у порівнянні з текстовими форматами, такими як XML та JSON, однак останні неефективні для серіалізації структурованих даних, тож в якості компромісу пропонуються бінарні дані з нестрогою схемою.

Авторами [4] розглянуто окремі методи серіалізації: Protobuf, Kryo, Protostuff (підтримує формати JSON, XML, YAML, KVP) та Jackson (пропонує три варіанти обробки JSON). Аналіз отриманих залежностей часу серіалізації/десеріалізації від кількості об’єктів показав, що за цим критерієм найбільш ефективним є метод Protobuf.

Окремими авторами [5] запропоновано використання власного формату серіалізації та механізму десеріалізації даних, наприклад, для врахуванням специфіки сутностей у реляційній моделі та підвищення продуктивності роботи програмного забезпечення, що потребувало великої роботи з побудови концепції експериментального формату.

Дослідження закордонних авторів присвячені в основному порівнянню XML, JSON з іншими текстовими серіалізаторами.

p-ISSN: 2225-6733; e-ISSN: 2519-271X

Автори [6] виділяють серіалізатори XML та JSON як найвідоміші формати текстових даних, порівнюючи їх з відносно новими форматами двійкової серіалізації ProtoBuf та Thrift.

У роботах, присвячених сфері Інтернету речей (IoT), проведено, наприклад, порівняльне дослідження форматів JSON та EXI [7], рекомендованих технічними специфікаціями ETSI M2M, а також нового формату саме для цих середовищ – PSON [8], який дозволяє спростити завдання серіалізації/десеріалізації та мінімізувати кількість повідомлень.

У роботі [9] проведено дослідження 12 бібліотек серіалізації об'єктів у форматах XML, JSON та у бінарних форматах, а в якості ефективності роботи оцінювали розмір серіалізованих файлів та час обробки. Проведений аналіз дозволив зробити висновки, що застосування кожної окремої бібліотеки доцільно в тому контексті, в якому вона була розроблена, що потребує індивідуального підходу в рамках конкретних проєктів.

Приклад такого дослідження наведено у статті [10], авторами якої розроблено ізольоване від решти операційної системи середовище для тестування, яке є відкритим та автоматизованим. Досліджувалися формати серіалізації XML, JSON, MessagePack, Avro, буфери протоколів, а також власний формат серіалізації і власні серіалізації. Процеси серіалізація та десеріалізація протестовано на PHP, Java та JavaScript з використанням 49 різних офіційних і сторонніх бібліотек, що дозволило виявити велику різницю в часі обробки між бібліотеками.

У даній роботі прийнято рішення провести дослідження серіалізаторів мови програмування C#, що не знайшло відображення у розглянутих вище дослідженнях.

На основі проведеного аналізу публікацій було виокремлено наступні популярні формати серіалізації даних: XML, JSON, BSON, MessagePack, Smile, Protocol Buffers, Flat Buffers, Apache Thrift, характеристику яких наведено нижче.

XML (Extensible Markup Language) – формат, який описує клас об'єктів даних, що називаються XML-документами, і частково описує поведінку комп'ютерних програм, які їх обробляють.

JSON (JavaScript Object Notation) – текстовий незалежний формат, який використовує конвенції мов програмування сімейства C. JSON побудований на двох структурах: колекції пар «назва/значення» та упорядковані списки значень, що реалізуються як масив, вектор, список або послідовність.

BSON (Binary JSON) – як і JSON, BSON підтримує багаторівневі структури документів та масивів. BSON можна порівнювати з бінарними форматами обміну, наприклад, Protocol Buffers. На відміну від останнього, він більш гнучкий. Однак BSON має накладні витрати, адже схема даних передається разом із самими даними.

Smile – бінарний формат даних, еквівалент стандартного формату даних JSON. Дані кодуються по секціях. Кожна секція складається з набору токенів, які формують відповідні ключі та значення.

MessagePack – механізм серіалізації/десеріалізації об'єктів та формат обміну даними, подібний до BSON та Smile. В MessagePack є своя система типів та присутні оптимізації для складних об'єктів та бінарних/текстових даних довільної довжини.

Protocol Buffers – нейтральний та платформо-незалежний спосіб серіалізації та передачі структурованих даних, розроблений Google. Кожне повідомлення в Protocol Buffers є невеликим логічним записом інформації, що містить серію пар «ім'я-значення».

Flat Buffers – формат та метод серіалізації даних, подібний до Protocol Buffers, але спеціально розроблений для систем із дійсно високим навантаженням. Його особливість полягає в оптимізованому використанні системних ресурсів – як часу на серіалізацію/десеріалізацію, так і на розмір використовуваної пам'яті. Flat Buffers надає доступ до даних без процесу серіалізації/десеріалізації.

Apache Thrift – механізм серіалізації з нижчими накладними витратами на відміну від таких альтернатив, як SOAP, за рахунок використання двійкового формату.

Серед наведеного переліку вирішено зосередити увагу на двох найпопулярніших форматах серіалізації даних – XML та JSON (бінарна серіалізація не розглядалася через виявлені вразливості десеріалізації для мови C#), при цьому слід зауважити, що кожний з них має декілька реалізацій із різними перевагами та недоліками. Цим обумовлена проблема із вибором конкретного протоколу, що буде використовуватись при написанні програмного забезпечення.

p-ISSN: 2225-6733; e-ISSN: 2519-271X

Порівняльний аналіз форматів XML та JSON дозволив виявити наступні спільні властивості: зрозумілість як для комп'ютера, так і для людини (наявна можливість редагування); широке застосування й підтримка у багатьох мовах програмування; формати є текстовими за відображенням структури даних і мають ієрархічну структуру даних, у якій кожне поле має своє ім'я та значення.

Перевагою XML у порівнянні JSON є підтримка додаткових типів даних, таких як логічні масиви, дати, зображення та простори імен.

Натомість можна виділити цілий ряд переваг JSON у порівнянні з XML, які полягають в наступному: JSON є більш простим та гнучким форматом серіалізації, має менший розмір файлів, використовує менше пам'яті й забезпечує більш швидку передачу даних. JSON – це засіб простого та компактного зберігання даних, його легше читати та редагувати. XML може зберігатись тільки у вигляді текстових файлів, у той час як JSON окрім текстового файлу може використовуватись разом з JavaScript. JSON відмінно підходить для обміну інформацією між програмними додатками. JSON більш поширений, тому не виникає проблем з кросплатформною сумісністю. Суттєвим аргументом на користь JSON є більший рівень безпеки, ніж у XML.

Отже, проаналізувавши наведені відмінності XML та JSON форматів, у даній роботі було вирішено дослідити саме JSON-серіалізацію, що обумовлено її актуальністю, поширеним використанням та необхідним рівнем безпеки.

Аналіз найбільш популярних програмні рішень для серіалізації об'єктів C# показав доцільність подальшого тестування таких серіалізаторів, як Jil, Json.NET (Newtonsoft.Json), Utf8Json та SpanJson з метою виявлення переваг та недоліків їхнього використання для конкретних задач.

Виклад основного матеріалу. У даній статті представлено розроблене авторами програмне забезпечення для тестування серіалізаторів з використанням різних за обсягом і типом даних об'єктів, що використовуються у програмних рішеннях, написаних мовою програмування C#.

Програма-тестувальник дозволяє визначити параметри роботи різних серіалізаторів даних для формату JSON, а також порівняти їх ефективність при серіалізації різних об'єктів. Результати проведених досліджень дозволять обґрунтовувати вибір найбільш результативних серіалізаторів для умов окремих практичних проєктів відповідно до різних типів даних. Основним критеріями оцінки роботи серіалізаторів обрано час на серіалізацію та обсяг спожитої оперативної пам'яті.

Діаграма класів програми-тестувальника серіалізаторів (рис. 1) включає наступні класи:

– SerializationTestSmall – описує екземпляр об'єкту невеликого розміру, що містить розповсюджені типи даних та методи серіалізації, що застосовуються до цього екземпляру об'єкту;

– SerializationTestBig – даний клас описує екземпляр об'єкту великого розміру, що містить окрім розповсюджених типів даних, також складні типи, які самі по собі є класами (клас BirthDate, клас Address), та складні типи формату відображення часу (DateTime);

– SerializationTestLongInheritance – завданням цього класу є тестування ефективності серіалізації екземпляра об'єкту, який утворено від класу з великим ланцюгом наслідування (еріалізації такого екземпляру об'єкту вимагає від серіалізаторів створення об'єктного графу, який містить множину взаємозалежних об'єктів);

– SerializationTestSimpleArray – даний клас демонструє поведінку серіалізаторів при серіалізації типу даних числовий масив int[];

– SerializationTestSimpleArray2d – цей клас показує, на скільки зростає навантаження (уповільнюється робота) на серіалізатори при серіалізації двовимірного числового масиву int[,] на відміну від одновимірного, який описано у попередньому класі;

– SerializationTestSimpleArray3d – клас описує екземпляр об'єкту трьохвимірного масиву int[,,], також демонструє різницю швидкодії порівняно з двовимірним та одновимірним масивами;

– SerializationTestDictionary – описує екземпляр об'єкту словник, що складається з пар «ключ – значення», типів «рядок – число» (string, int).

Кожен з вищеописаних класів створює відповідний екземпляр об'єкту. Безпосередньо класи, що описують об'єкти, потрібні кожному класу для створення екземпляру, описано окремо: SmallClass, BiglClass, Developer, SimpleArrayClass, Array2dClass, DictionaryClass.

p-ISSN: 2225-6733; e-ISSN: 2519-271X

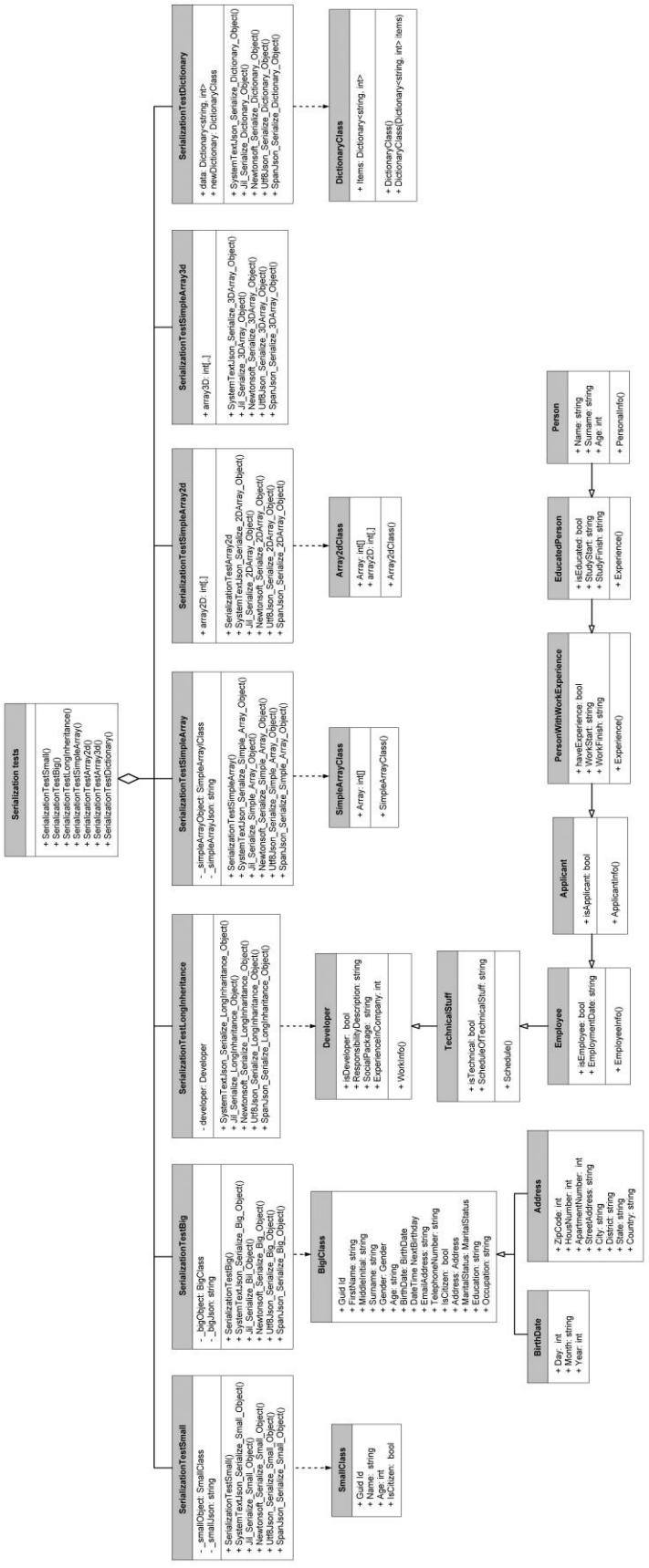

Рис. 1 – Діаграма класів програми-тестувальника серіалізаторів

p-ISSN: 2225-6733; e-ISSN: 2519-271X

У рамках даного дослідження для побудови діаграми компонентів програми-тестувальника серіалізаторів (рис. 2) використано підключені програмні пакети, необхідні для коректного проведення експериментів.

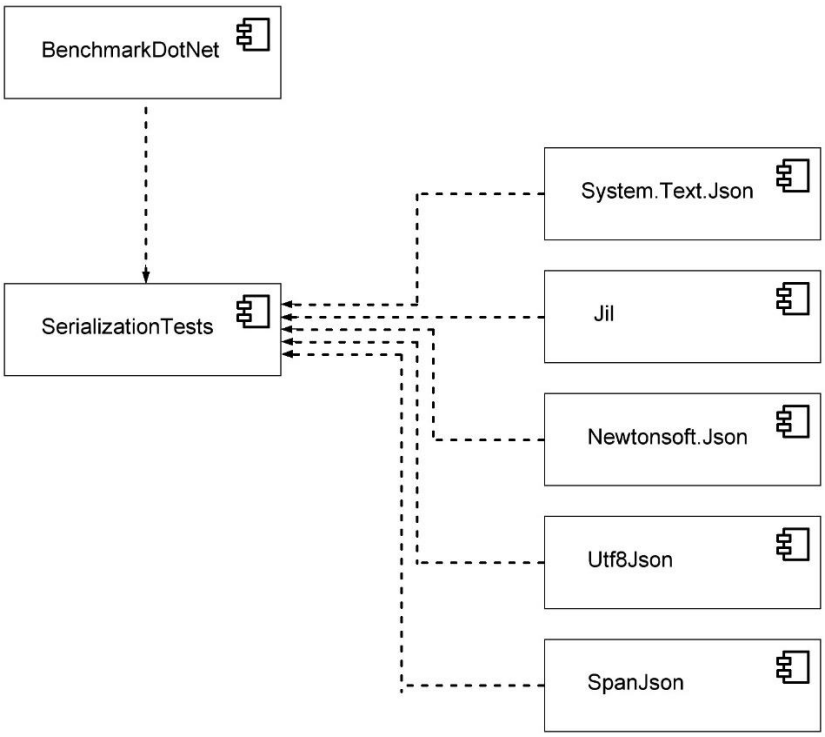

Рис. 2 – Діаграма компонентів програми-тестувальника серіалізаторів

Розробка програмного забезпечення здійснювалася з використанням мови програмування C#, а в якості середовища розробки було обрано Microsoft Visual Studio. Програмні пакети, зображені на діаграмі, підключали за допомогою менеджера пакетів NuGet.

Виконання програми складається з 7 етапів, кожен з яких є окремим експериментом тестування серіалізаторів на вказаному типі даних. Кожний етап представляє собою створення екземпляру класу, який буде використовуватись як вхідні дані для серіалізації. Після створення об’єкту (він же екземпляр класу) його послідовно серіалізують 5 серіалізаторів, які було обрано для участі в експерименті (вони є окремими програмними пакетами і вимагають підключення для застосування в проєкті):

– System.Text.Json;
– Jil;
– Newtonsoft.Json;
– Utf8Json;
– SpanJson.

Фреймворк BenchmarkDotNet здійснює послідовні навантаження на кожний з серіалізаторів та формує звіт результатів порівняння серіалізаторів у зручному вигляді.

Схема взаємодії програмних модулів, показана на рис. 3, зображує послідовність виконання запланованих експериментів, які відрізняються між собою типами даних та обсягом навантаження.

p-ISSN: 2225-6733; e-ISSN: 2519-271X

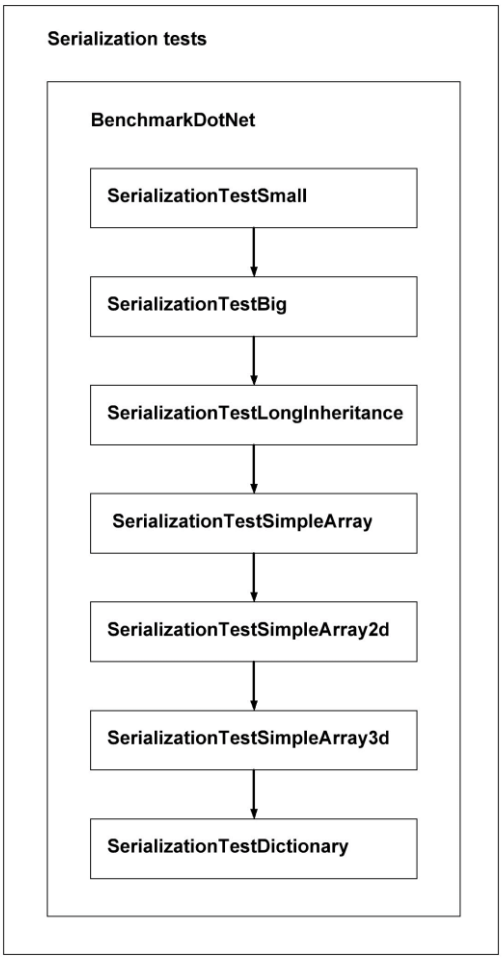

Рис. 3 – Схема взаємодії програмних модулів тестувальника серіалізаторів

Приклад роботи програми-тестувальника Serialization tests наведено на рис. 4.

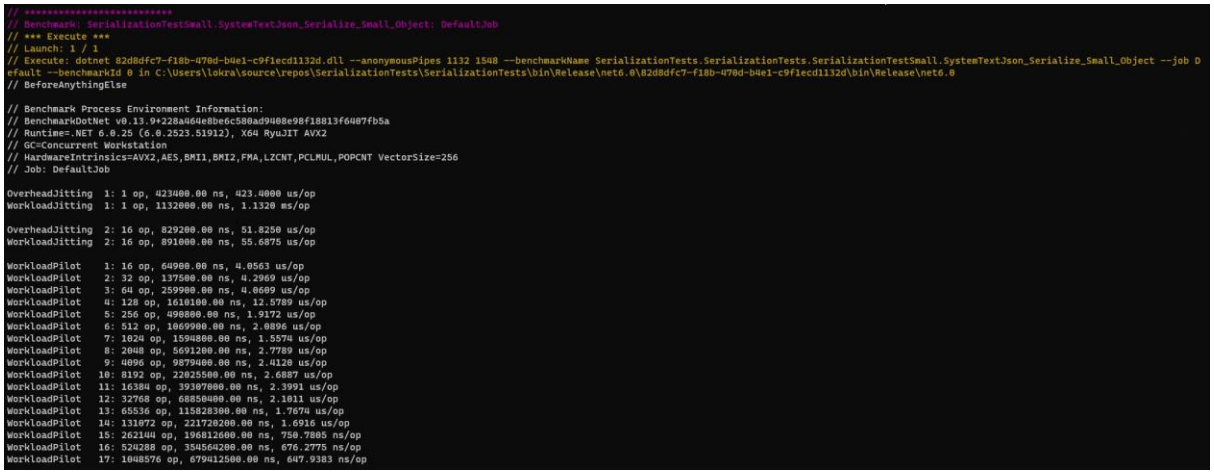

Рис. 4 – Запуск виконання тестів серіалізаторів

На рис. 5 наведено інформацію про стан поточних операцій тестування: кількість виконаних операції, час виконання операцій (в наносекундах) та час виконання однієї операції. Тестування здійснювалося на різних обсягах даних.

p-ISSN: 2225-6733; e-ISSN: 2519-271X

```
OverheadJitting  1: 1 op, 423400.00 ns, 423.4000 us/op
WorkloadJitting  1: 1 op, 1132000.00 ns, 1.1320 ms/op

OverheadJitting  2: 16 op, 829200.00 ns, 51.8250 us/op
WorkloadJitting  2: 16 op, 891000.00 ns, 55.6875 us/op

WorkloadPilot    1: 16 op, 64900.00 ns, 4.0563 us/op
WorkloadPilot    2: 32 op, 137500.00 ns, 4.2969 us/op
WorkloadPilot    3: 64 op, 259900.00 ns, 4.0609 us/op
WorkloadPilot    4: 128 op, 1610100.00 ns, 12.5789 us/op
```

Рис. 5 – Інформація про стан поточних операцій тестування

Після закінчення кожного з 7 експериментів формується звіт (рис. 6), який містить технічні та програмні характеристики системи, на базі якої було проведено тестування, а також сформовану таблицю з результатами роботи серіалізаторів.

```
// * Summary *

BenchmarkDotNet v0.13.9+228a464e8be6c580ad9408e98f18813f6407fb5a, Windows 11 (10.0.22621.2715/22H2/2022Update/SunValley2)
Intel Pentium Gold 7505 2.00GHz, 1 CPU, 4 logical and 2 physical cores
.NET SDK 8.0.100
  [Host]     : .NET 6.0.25 (6.0.2523.51912), X64 RyuJIT AVX2
  DefaultJob : .NET 6.0.25 (6.0.2523.51912), X64 RyuJIT AVX2


| Method                                | Mean       | Error    | StdDev   | Rank | Gen0   | Allocated |
|-------------------------------------- |-----------:|---------:|---------:|-----:|-------:|----------:|
| SystemTextJson_Serialize_Small_Object |   640.6 ns |  5.34 ns |  5.00 ns |    4 | 0.3519 |     736 B |
| Jil_Serialize_Small_Object            |   385.2 ns |  7.30 ns |  7.50 ns |    3 | 0.4663 |     976 B |
| Newtonsoft_Serialize_Small_Object     | 1,151.5 ns | 22.59 ns | 30.91 ns |    5 | 0.8869 |    1856 B |
| Utf8Json_Serialize_Small_Object       |   274.6 ns |  5.31 ns |  5.90 ns |    2 | 0.1144 |     240 B |
| SpanJson_Serialize_Small_Object       |   205.1 ns |  3.58 ns |  3.35 ns |    1 | 0.1147 |     240 B |

// * Hints *
Outliers
  SerializationTestSmall.Jil_Serialize_Small_Object: Default -> 1 outlier  was  removed, 3 outliers were detected (367.65 ns, 371.79 ns, 402.14 ns)

// * Legends *
  Mean      : Arithmetic mean of all measurements
  Error     : Half of 99.9% confidence interval
  StdDev    : Standard deviation of all measurements
  Rank      : Relative position of current benchmark mean among all benchmarks (Arabic style)
  Gen0      : GC Generation 0 collects per 1000 operations
  Allocated : Allocated memory per single operation (managed only, inclusive, 1KB = 1024B)
  1 ns      : 1 Nanosecond (0.000000001 sec)

// * Diagnostic Output - MemoryDiagnoser *


// ***** BenchmarkRunner: End *****
Run time: 00:01:49 (109.49 sec), executed benchmarks: 5

Global total time: 00:02:08 (128.61 sec), executed benchmarks: 5
```

Рис. 6 – Звіт про проведений експеримент з результатами ефективності серіалізаторів

Таблиця звіту проведеного експерименту (див. рис. 6) містить наступну інформацію (опис параметрів наведено у вікні після таблиці):

– Method – перелік серіалізаторів, які брали участь в експерименті із зазначенням, об’єкт якого класу вони серіалізують;

– Mean – середнє арифметичне швидкості серіалізації при роботі з конкретним об’єктом (в наносекундах);

– Error – половина від 99,9% довірчого інтервалу;

– StdDev – середньо квадратичне відхилення при вимірюванні;

– Rank – позиція відносно інших серіалізаторів за результатами всіх проведених вимірювань;

– Gen0 – використана генерація пам’яті;

– Allocated –пам’ять, виділена за одну операцію (в байтах).

Після таблиці також зазначається час, витрачений на проведення експерименту.

p-ISSN: 2225-6733; e-ISSN: 2519-271X

Підсумкову таблицю, яка відображає результати всіх проведених експериментів, наведено на рис. 7.

Method	Mean	Error	StdDev	Rank	Gen0	Allocated
Serialize_Small_Object						
SystemTextJson	640.6 ns	5.34 ns	5.00 ns	4	0.3519	736 B
Jil	385.2 ns	7.30 ns	7.50 ns	3	0.4663	976 B
Newtonsoft	1,151.5 ns	22.59 ns	30.91 ns	5	0.8869	1856 B
Utf8Json	274.6 ns	5.31 ns	5.90 ns	2	0.1144	240 B
SpanJson	205.1 ns	3.58 ns	3.35 ns	1	0.1147	240 B
Serialize_Big_Object						
SystemTextJson	135.39 ns	2.201 ns	2.059 ns	4	0.0880	184 B
Jil	73.96 ns	1.461 ns	2.559 ns	3	0.1109	232 B
Newtonsoft	195.98 ns	3.746 ns	7.568 ns	5	0.5584	1168 B
Utf8Json	47.37 ns	0.863 ns	0.720 ns	1	0.0153	32 B
SpanJson	50.78 ns	0.288 ns	0.241 ns	2	0.0153	32 B
Serialize_LongInheritance_Object						
SystemTextJson	1,249.0 ns	22.18 ns	18.52 ns	4	0.5226	1096 B
Jil	558.4 ns	10.95 ns	19.47 ns	1	0.6227	1304 B
Newtonsoft	2,772.0 ns	44.26 ns	41.40 ns	5	1.4381	3008 B
Utf8Json	692.0 ns	13.54 ns	18.54 ns	3	0.4511	944 B
SpanJson	611.7 ns	11.95 ns	17.52 ns	2	0.4511	944 B
Serialize_Simple_Array_Object						
SystemTextJson	639.2 ns	8.21 ns	7.68 ns	4	0.3014	632 B
Jil	420.3 ns	4.32 ns	4.04 ns	3	0.2751	576 B
Newtonsoft	1,786.1 ns	21.92 ns	20.51 ns	5	0.8831	1848 B
Utf8Json	214.4 ns	2.09 ns	1.96 ns	2	0.0648	136 B
SpanJson	163.8 ns	2.91 ns	2.73 ns	1	0.0648	136 B
Serialize_2DArray_Object						
SystemTextJson	NA	NA	NA	?	NA	NA
Jil	77.44 ns	1.518 ns	2.495 ns	1	0.1109	232 B
Newtonsoft	3,130.10 ns	40.901 ns	38.259 ns	4	1.0796	2264 B
Utf8Json	336.13 ns	4.618 ns	4.094 ns	3	0.0687	144 B
SpanJson	273.11 ns	5.305 ns	5.448 ns	2	0.0687	144 B
Serialize_3DArray_Object						
SystemTextJson	NA	NA	NA	?	NA	NA
Jil	76.86 ns	1.506 ns	2.558 ns	1	0.1109	232 B
Newtonsoft	8,667.72 ns	165.211 ns	190.257 ns	3	1.9836	4176 B
Utf8Json	983.45 ns	6.491 ns	6.072 ns	2	0.1907	400 B
SpanJson	NA	NA	NA	?	NA	NA
Serialize_Dictionary_Object						
SystemTextJson	1,438.7 ns	10.08 ns	9.43 ns	3	0.4616	968 B
Jil	1,486.1 ns	21.51 ns	20.12 ns	4	0.6886	1440 B
Newtonsoft	2,875.2 ns	52.70 ns	49.29 ns	5	1.1787	2472 B
Utf8Json	1,188.2 ns	23.13 ns	22.72 ns	2	0.2251	472 B
SpanJson	993.3 ns	19.45 ns	30.29 ns	1	0.2518	528 B

Рис. 7 – Підсумкова таблиця результатів усіх експериментів

Результати роботи програми для тестування серіалізаторів при серіалізації маленького об’єкта наведено на рис. 8. Даний об’єкт складається з 4 полів та не несе великого навантаження

p-ISSN: 2225-6733; e-ISSN: 2519-271X

на серіалізатори, а результати використовувалися для розуміння стандартної швидкості серіалізації і споживання ресурсів при порівнянні з серіалізацією інших об'єктів.

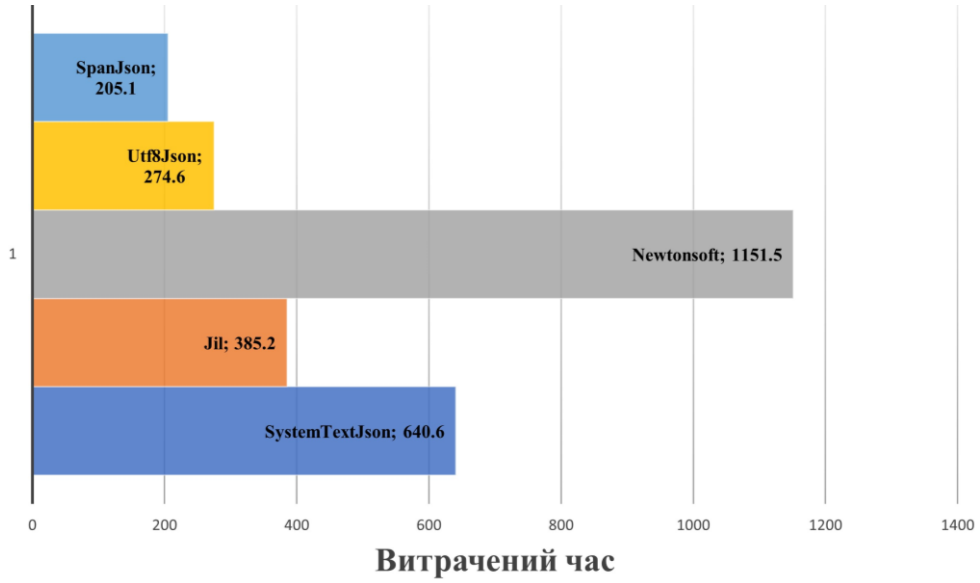

Рис. 8 – Діаграма витрат часу при серіалізації маленького об'єкта

При серіалізації маленького об'єкта стандарт індустрії Newtonsoft.Json, який рекомендується компанією Microsoft та є найбільш широко поширеним серіалізатором, має за всіма параметрами в 2-3 рази гірші показники, ніж інші серіалізатори, що брали участь в експерименті. Newtonsoft.Json отримує 5 місце за результатами ефективності. Серіалізатор System.Text.Json, який за замовчанням присутній в стандартних бібліотеках C#, продемонстрував середні результати за швидкістю та велику похибку вимірювання та споживання ресурсів. BenchmarkDotNet присвоює йому 4 місце. SpanJson показав найкращі результати за всіма параметрами, тому посідає 1 місце у цьому експерименті. Utf8Json та Jil показали високу результативність та знаходяться на 2 і 3 місцях відповідно.

Результати роботи програми при серіалізації великого об'єкта зображено на рис. 9.

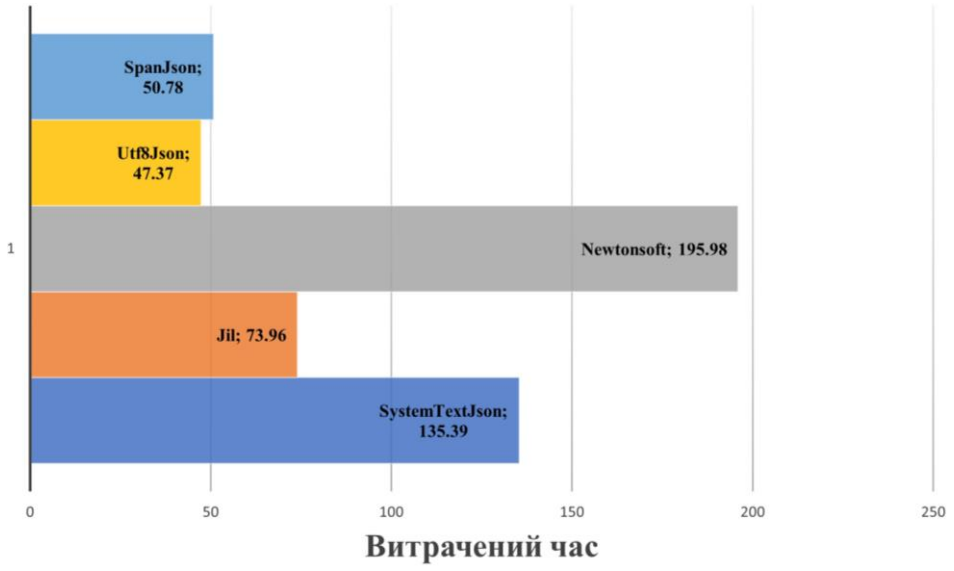

Рис. 9 – Діаграма витрат часу при серіалізації великого об'єкта

p-ISSN: 2225-6733; e-ISSN: 2519-271X

Як видно з діаграми, розмір об’єкта вплинув лише на показники Utf8Json – він найкраще за всі інші серіалізатори впорався з завданням. Це дозволяє зробити висновки, що при серіалізації великих об’єктів слід віддавати перевагу використанню серіалізатора Utf8Json. Для інших серіалізаторів показники швидкості та споживання ресурсів зберегли таку саму пропорцію, як і при серіалізації маленького об’єкта, тобто кількість даних в об’єкті не має впливу на змінення швидкості серіалізації.

Слід зауважити, що при проведені експериментів, результати яких відображено на рис. 8 та рис. 9, кількість маленьких об’єктів, згенерованих фреймворком BenchmarkDotNet, була значно більшою, ніж кількість великих об’єктів, що вплинуло на середні арифметичні значення показників.

Результати роботи програми тестування серіалізаторів при серіалізації об’єкта зі складним ланцюгом спадкування класів зображено на рис. 10. Особливістю цього об’єкта є клас, який його описує: він має складну систему спадкування, отже, для створення об’єкта такого класу потрібно заповнити усі поля класів, від яких було успадкувано фінальний клас.

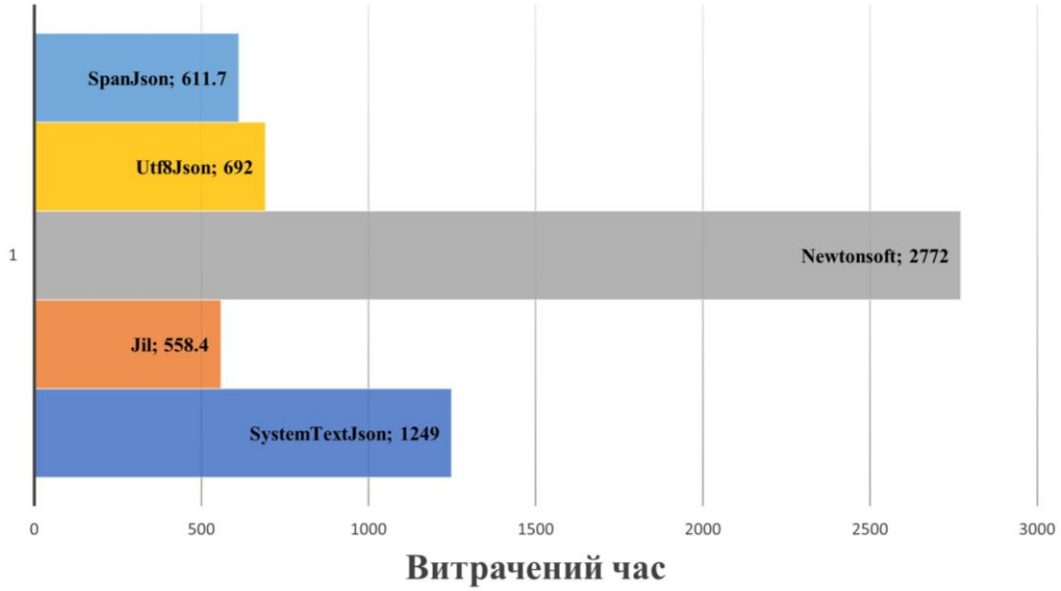

Рис. 10 – Діаграма витрат часу при серіалізації об’єкта зі складним ланцюгом спадкування класів

Аналіз результатів показує, що серіалізацію об’єктів, утворених від класів зі складним ланцюгом спадкування, краще всього виконує Jil серіалізатор, який є лідером за швидкістю, хоча й показує середні показники споживання ресурсу пам’яті. SpanJson та Utf8Json мають практично однакові результати і лише трохи поступаються Jil у цьому експерименті. System.Text.Json витратив значно більше часу, ніж три лідери експерименту. Newtonsoft.Json має найгірші показники та значно відстає від інших серіалізаторів за всіма параметрами.

Отже, якщо проєкт вимагає серіалізації об’єктів, створених від класів зі складним ланцюгом спадкування, варто звернути увагу на Jil серіалізатор, який значно краще впорався з завданням у даному експерименті.

Результати тестування при серіалізації об’єкта, що містить масив натуральних чисел, наведено на рис. 11.

Пропорції показників при серіалізації об’єкта, що містить масив натуральних чисел, схожі з результатами серіалізації маленького об’єкта. Серіалізатори SpanJson, Utf8Json та Jil знову демонструють найкращі показники ефективності. Також прийнятні результати швидкості та використання пам’яті має стандартний серіалізатор System.Text.Json. Як і в попередніх експериментах, Newtonsoft.Json показує значне відставання від своїх конкурентів, хоча й є одним з найпоширеніших серіалізаторів і часто використовується в практичних проєктах.

p-ISSN: 2225-6733; e-ISSN: 2519-271X

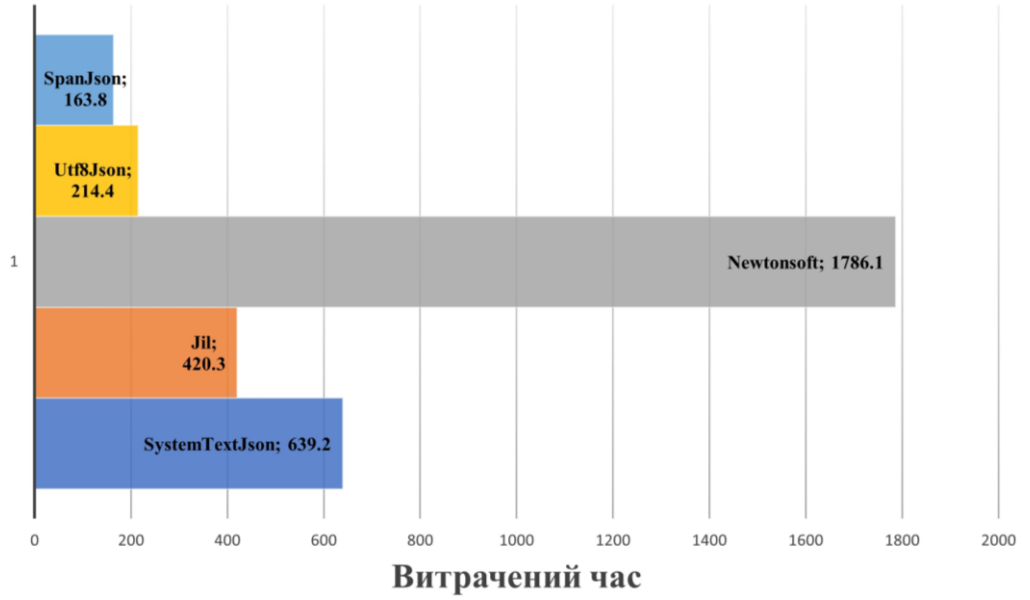

Рис. 11 – Діаграма витрат часу при серіалізації об’єкта, що містить масив натуральних чисел

Результати роботи програми-тестувальника при серіалізації об’єкта, що містить двовимірний масив натуральних чисел, зображено на рис. 12.

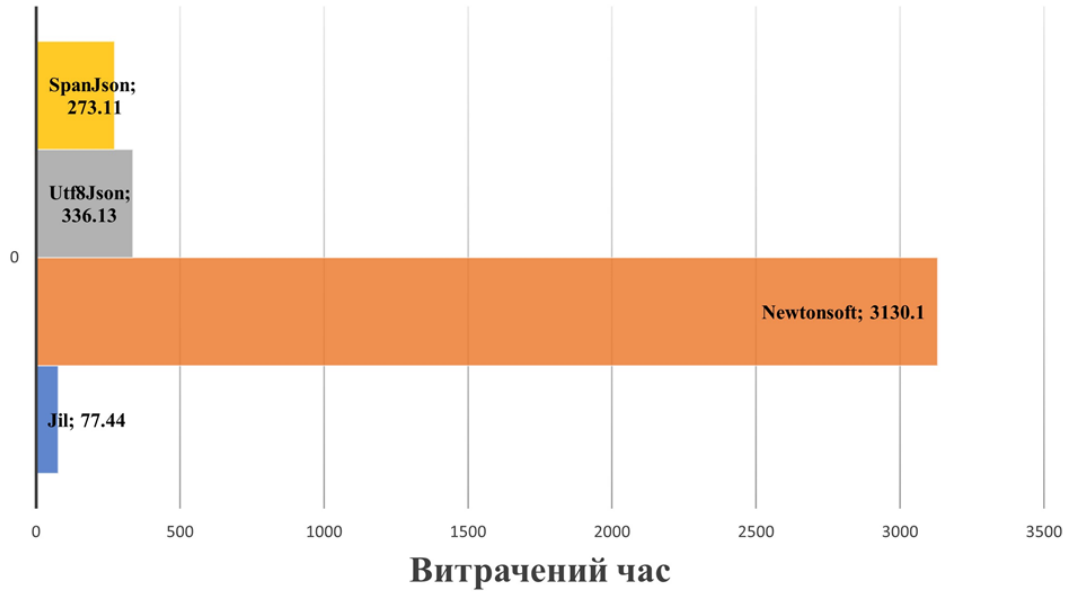

Рис. 12 – Діаграма витрат часу при серіалізації об’єкта, що містить двовимірний масив натуральних чисел

Двомірні масиви менше розповсюджені в практичних проєктах, але представляють інтерес для експериментів. На діаграмі відсутні результати серіалізації System.Text.Json, тому що аналіз помилок, які виникли в результаті експерименту, дозволив зробити висновок, що System.Text.Json не працює з типом даних двовимірний масив та генерує відповідне виключення за результатами експерименту з двовимірними масивами. Найкраще з таким типом навантажень впорався Jil серіалізатор. Серіалізатори SpanJson та Utf8Json також продемонстрували хороші результати швидкості серіалізації та використання пам’яті. Newtonsoft.Json знову показав значне

p-ISSN: 2225-6733; e-ISSN: 2519-271X

відставання від своїх конкурентів, але, на відміну від серіалізатора System.Text.Json, він впорався з завданням серіалізації двовимірного масива.

Експеримент з серіалізації об’єкта, що містить тривимірний масив натуральних чисел, демонструє незвичайний результат, що зображено на рис. 13.

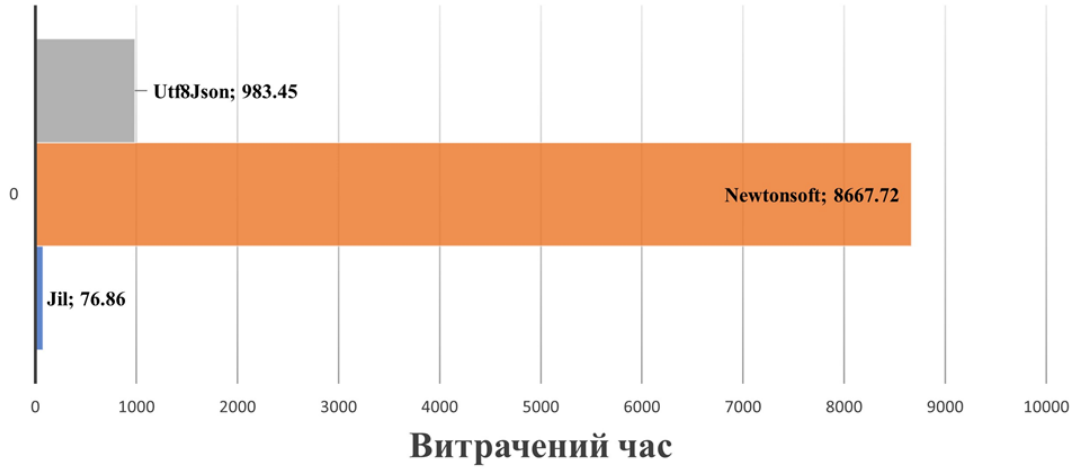

Рис. 13 – Діаграма витрат часу при серіалізації об’єкта, що містить тривимірний масив натуральних чисел

Як і у випадку з двовимірним масивом, серіалізатор System.Text.Json згенерував виключення та не брав участь в експерименті. При цьому разом з ним у випадку з тривимірним масивом відповідне виключення згенерував серіалізатор SpanJson.

Використання тривимірних масивів – це рідкісний практичний випадок, проте такий спосіб зберігання даних можливий. Аналіз результатів показав, що Jil серіалізатор знову демонструє найкращі показники, що на порядок вищі за показники тих його конкурентів, які змогли провести серіалізацію тривимірного масива. Utf8Json показав хороші результати швидкості серіалізації. Newtonsoft.Json знову значно відстає, але впорався з цією серіалізацією.

Результати роботи програми для тестування при серіалізації об’єкта, що містить словник, зображені на рис. 14.

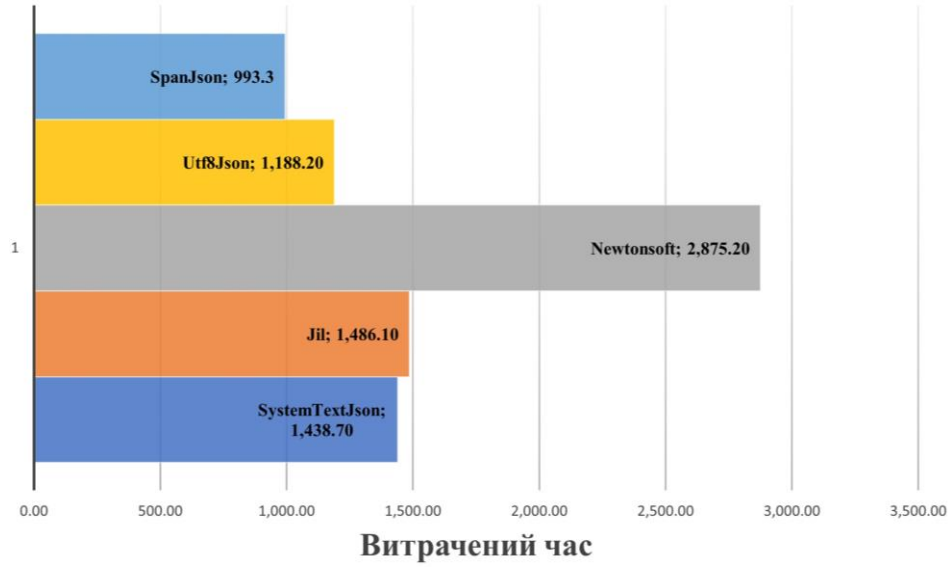

Рис. 14 – Діаграма витрат часу при серіалізації об’єкта, що містить словник

Використання словників для зберігання даних – це поширена практика, тому такий експеримент є необхідним. Аналіз отриманих результатів показав, що з цим завдання краще за конкурентів впорався серіалізатор SpanJson. Utf8Json теж демонструє хороші показники, через що BenchmarkDotNet присвоює йому 2 місце. Стандартний серіалізатор System.Text.Json теж демонструє дуже добрі результати порівняно з його можливостями у попередніх експериментах. Jil серіалізатор навпаки, на відміну від якісних результатів у попередніх експериментах, має низькі показники, через що посідає тільки 4 місце при серіалізації об'єкта, який містить словник.

На основі отриманих результатів роботи програми-тестувальника при реалізації серіалізації різних об'єктів, можна зробити висновок, що не існує універсального серіалізатора, який буде показувати найкращі результати в усіх випадках. Тому при реалізації окремого програмного рішення для виконання серіалізації доцільно використовувати різні технології. Це вимагає у кожному випадку проведення нових експериментальних досліджень для конкретних об'єктів, які будуть безпосередньо використовуватись в проєкті. Також пильну увагу необхідно звертати на особливості та специфіку використання того чи іншого серіалізатора. Крім того, при зміні версій та коду продуктів або появі нових рішень серіалізації потрібно проводити додаткові дослідження для кожного конкретного випадку використання різних об'єктів.

Висновки

Таким чином, розроблено програму для тестування серіалізаторів з використанням різних за обсягом і типом даних об'єктів. Аналіз найбільш популярних програмні рішень для серіалізації об'єктів C# показав доцільність використання JSON формату та проведення тестування таких серіалізаторів, як Jil, Json.NET (Newtonsoft.Json), Utf8Json та SpanJson, з метою виявлення переваг та недоліків їхнього використання для реалізації конкретних задач та проєктів.

Результати експериментальних досліджень, отримані в рамках даної роботи, показали залежність ефективності серіалізаторів від типу та обсягу даних, які потрібно серіалізувати.

Кожна окрема технологія серіалізації ефективно здійснює серіалізацію одних об'єктів та поступається показниками при використанні інших об'єктів. Серіалізатор SpanJson показав найкращі результати при роботі з об'єктами невеликого розміру та об'єктами, що містять одновимірні масиви та словники. Utf8Json краще за конкурентів впорався з серіалізацією об'єктів великого обсягу. Серіалізатор Jil показав найкращій результат при використанні об'єктів зі складними ланцюгами спадкування класів. Стандартний серіалізатор System.Text.Json показав середні результати при виконанні експериментів. Newtonsoft.Json виявився найповільнішим та споживав більше системних ресурсів, хоча й є одним з найпоширеніших в практичних проєктах.

За допомогою проведених експериментів на практичних прикладах було доведено необхідність тестування серіалізаторів для виявлення найкращого для кожного практичного проєкту, який вимагає використання серіалізації.

Перелік використаних джерел:

1. Грудзинський Ю.Є., Марков Р.В. Вибір протоколу серіалізації при розробці програмного забезпечення. *Вісник НТУ «ХПІ». Серія: Нові рішення в сучасних технологіях*. 2016. № 12(1184). С. 13-16. DOI: https://doi.org/10.20998/2413-4295.2016.12.15.
2. Андрущенко Р. Порівняльний аналіз показників ефективності методів серіалізації даних у компютерних мережах. *Технічні науки та технології*. 2019. № 1(15). С. 115-126. DOI: https://doi.org/10.25140/2411-5363-2019-1(15)-115-126.
3. Андрущенко Р.Б., Зайцев С.В., Солдатов А.Ю. Аналіз методів серіаліазції структурованих даних для передачі в протоколах прикладного рівня моделі OSI. *Математичне моделювання в економіці*. 2018. № 3. С. 52-70. URL: http://dspace.nbuv.gov.ua/handle/123456789/162043.
4. Попенко Д.В., Курдеча В.В. Аналіз методів серіалізації об'єктів для побудови платформи великих індустріальних даних. *Перспективи телекомунікацій* : збірник матеріалів Міжнародної науково-технічної конференції, м. Київ, 13-17 квіт. 2020 р. С. 1-3. URL: http://conferenc.its.kpi.ua/proc/article/view/201701.
5. Літвінова Н., Альперт М., Погульський А. Підвищення ефективності обміну даними сутностей у реляційному представленні та їх обробки. *Технічні науки та технології*. 2021. № 1(23). С. 81-86. DOI: https://doi.org/10.25140/2411-5363-2021-1(23)-81-86%20.

6. Sumaray A., Makki S.K. A comparison of data serialization formats for optimal efficiency on a mobile platform. *6th International Conference on Ubiquitous Information Management and Communication (ICUIMC'12)*, Kuala Lumpur, Malaysia, 20-22 February 2012. Vol. 48. Pp. 1-6. DOI: https://doi.org/10.1145/2184751.2184810.
7. Performance analysis of data serialization formats in m2m wireless sensor networks / F. Pacini, F.A. Aderohunmu, A. Azzarà, S. Bocchino, P. Pagano, M. Petracca. *European Conference on Wireless Sensor Networks*, Porto, Portugal, 9-11 February 2018. Pp. 7-8. URL: https://cister-labs.pt/docs/poster_demo_session_proceedings_of_the_12th_european_conference_on_wireless_sensor_networks_(ewsn_15)/1075/view.pdf#page=8.
8. PSON: A Serialization Format for IoT Sensor Networks / Luis Á., Casares P., Cuadrado-Gallego J.J., Patricio M.A. *Sensors*. 2021. Vol. 23(13). Pp. 1-18. DOI: https://doi.org/10.3390/s21134559.
9. Kazuaki M. Performance evaluation of object serialization libraries in XML, JSON and binary formats. *Second International Conference on Digital Information and Communication Technology and it's Applications (DICTAP)*, Bangkok, Thailand, 16-18 May 2012. Pp. 177-182. DOI: https://doi.org/10.1109/DICTAP.2012.6215346.
10. Vanura J., Kriz P. Perfomance Evaluation of Java, JavaScript and PHP Serialization Libraries for XML, JSON and Binary Formats. *Services Computing – SCC 2018*: 15th International Conference, Seattle, USA, 25-30 June 2018. 2018. Pp. 166-175. DOI: https://doi.org/10.1007/978-3-319-94376-3_11.

References:

1. Yu.Ie. Grudzynskyy, and R.V. Markov, «Vybir protokolu serializatsii pry rozrobtsi prohramnoho zabezpechennia» [«Protocol selection for serialization software development communication module SCADA-systems»], *Visnyk NTU «KhPI». Seriia: Novi rishennia v suchasnykh tekhnolohiiakh – Bulletin of the National Technical University «KhPI» Series: New solutions in modern technologies,* № 12(1184), pp. 13-16, 2016. **doi: 10.20998/2413-4295.2016.12.15**. (Ukr.)
2. R. Andrushchenko, «Porivnialnyi analiz pokaznykiv efektyvnosti metodiv serializatsii danykh u kompiuternykh merezhakh» [«Comparative analysis of the performance characteristics of data serialization methods in computer networks»], *Tekhnichni nauky ta tekhnolohii – Technical sciences and technologies,* № 1(15), pp. 115-126, 2019. **doi: 10.25140/2411-5363-2019-1(15)-115-126**. (Ukr.)
3. R.B. Andrushchenko, S.V. Zaitsev, and A.Iu. Soldatov, «Analiz metodiv serialiaztsii strukturovanykh danykh dlia peredachi v protokolakh prykladnoho rivnia modeli OSI» [«Analysis of structured data serialization methods for transmission in application layer protocols of the OSI model»], *Matematychne modeliuvannia v ekonomitsi – Mathematical modeling in economics*, № 3, pp. 52-70, 2018. Available: http://dspace.nbuv.gov.ua/handle/123456789/162043. (Ukr.)
4. D.V. Popenko, and V.V. Kurdecha, «Analiz metodiv serializatsii obiektiv dlia pobudovy platformy velykykh industrialnykh danykh» [«Analysis of object serialization methods for building a big industrial data platform»], in Perspektyvy telekomunikatsii : zbirnyk materialiv Mizhnarodnoi naukovo-tekhnichnoi konferentsii [Perspectives of telecommunications: a collection of materials of the International Scientific and Technical Conference], Kyiv, 2020, pp. 1-3. Available: http://conferenc.its.kpi.ua/proc/article/view/201701. (Ukr.)
5. N. Litvinova, M. Alpert, and A. Pohulskyi, «Pidvyshchennia efektyvnosti obminu danymy sutnostei u reliatsiinomu predstavlenni ta yikh obrobky» [«Improving the efficiency of data exchange of entities in relational representation and their processing»], *Tekhnichni nauky ta tekhnolohii – Technical sciences and technologies,* № 1(23), pp. 81-86, 2021. **doi: 10.25140/2411-5363-2021-1(23)-81-86%20**. (Ukr.)
6. A. Sumaray, and S.K. Makki, «A comparison of data serialization formats for optimal efficiency on a mobile platform», in 6th International Conference on Ubiquitous Information Management and Communication (ICUIMC'12), Kuala Lumpur, 2012, vol. 48, pp. 1-6. **doi: 10.1145/2184751.2184810**.
7. F. Pacini, F.A. Aderohunmu, A. Azzarà, S. Bocchino, P. Pagano, and M. Petracca, «Performance analysis of data serialization formats in m2m wireless sensor networks», in European Conference on Wireless Sensor Networks, Porto, 2018, pp. 7-8. Available: https://cister-

labs.pt/docs/poster_demo_session_proceedings_of_the_12th_european_conference_on_wireless_sensor_networks_(ewsn_15)/1075/view.pdf#page=8.
8. Á. Luis, P. Casares, J.J. Cuadrado-Gallego, and M.A. Patricio, «PSON: A Serialization Format for IoT Sensor Networks», *Sensors*, vol. 23(13), pp. 1-18, 2021. **doi: 10.3390/s21134559**.
9. M. Kazuaki, «Performance evaluation of object serialization libraries in XML, JSON and binary formats», in Second International Conference on Digital Information and Communication Technology and it's Applications (DICTAP), Bangkok, 2012, pp. 177-182. **doi: 10.1109/DICTAP.2012.6215346**.
10. J. Vanura, and P. Kriz, «Perfomance Evaluation of Java, JavaScript and PHP Serialization Libraries for XML, JSON and Binary Formats», in 15th International Conference Services Computing SCC 2018, Seattle, 2018, pp. 166-175. **doi: 10.1007/978-3-319-94376-3_11**.

Рецензент: О.І. Проніна
канд. техн. наук, доц., ДВНЗ «ПДТУ»

Стаття надійшла 05.07.2023
Стаття прийнята 13.08.2023

УДК 004.932:004.94 **doi: 10.31498/2225-6733.47.2023.299972**

© Тузенко О.О.[1], Володін С.І.[2]

ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ ДЛЯ ПОЛІПШЕННЯ ЯКОСТІ ЗОБРАЖЕННЯ ПРИ ЗБІЛЬШЕННІ ЙОГО ФОРМАТУ ПІСЛЯ ЗЙОМКИ

У статті розглянуто цифрові зображення різних типів, а також види комп'ютерної графіки. Виявлено, що в растровій графіці базовим компонентом зображення є точка, в такому випадку у векторній графіці – лінія. Різноманітні векторні формати мають різноманітні колірні можливості. Основною перевагою є можливість масштабування їх без втрати якості, а також відносно незначний розмір файлів, що їх містять. Це спрощує передачу векторних зображень по електронних каналах зв'язку. Запропоновано конкретний алгоритм поліпшення растрових зображень, а саме з використанням квадратичного кореня значень кольорів й без використанням квадратичного кореня значень кольорів. Основним завданням було досягнення результату вдосконалення випадкового кольорового растрового зображення низької роздільної здатності без втрати якості та роздільної здатності. Ключовим моментом методу є порівняння та поєднання вертикальної, горизонтальної та діагональної інтерполяції, що дозволяє досягти кращої точності обчислення глибини кольору. Цей метод ніколи не використовувався в комерційному науковому програмному забезпеченні, хоча існують різні варіанти комбінованих методів інтерполяції, подібних до поточного. У цій статті було досліджено два різні підходи до повторного розрахунку матриці зображення під час уточнення зображення, а саме його збільшення, як квадратичне значення глибини кольору вплине на цільове значення кольору. Результати показують, що такий підхід дозволяє зберегти більше деталей у тінях і контури під час інтерполяції, хоча зображення дещо втрачає глибину кольору. Експеримент показує, що цей метод інтерполяції з квадратичним коренем значень кольорів дозволяє збільшувати та покращувати кольорові зображення зі складною структурою тональної кривої та зберігати деталі об'єктів на місці, хоча глибина кольору погіршується, особливо в найглибших відтінках і в чорному. Навпаки, метод

[1] *канд. техн. наук, доцент, ДВНЗ «Приазовський державний технічний університет», м. Дніпро, ORCID: 0000-0002-4920-9417, tuzenkooa@gmail.com*
[2] *ст. викладач, ДВНЗ «Приазовський державний технічний університет», м. Дніпро, volodinsi57@gmail.com*