Науковий вісник НЛТУ України
Scientific Bulletin of UNFU
ISSN 1994-7836 (print)
ISSN 2519-2477 (online)
https://doi.org/10.36930/40360116
Article received: 18.12.2025, accepted: 06.02.2026
Article published: 26.02.2026
UDC 004.[02+4+67+9]
Corresponding author: Yu. Yu. Bilas yurii.y.bilas@lpnu.ua

Ю. Ю. Білас, В. В. Різник
Національний університет "Львівська політехніка", м. Львів, Україна

АЛГОРИТМ ПАРАЛЕЛЬНО-ПОСЛІДОВНОГО ЗАВАНТАЖЕННЯ ЦИФРОВИХ РЕСУРСІВ ІЗ КОНТРОЛЬОВАНОЮ КІЛЬКІСТЮ ОДНОЧАСНИХ ЗАВАНТАЖЕНЬ

Досліджено особливості організації процесу завантаження цифрових ресурсів у клієнтських веборієнтованих та гібридних мультимедійних застосунках за умов обмеженої кількості одночасних мережевих з'єднань. Проаналізовано наявні підходи до завантаження цифрових ресурсів, зокрема – послідовне, необмежене паралельне завантаження (усі водночас), стратегії з фіксованим рівнем паралелізму, а також стандартні механізми завантаження цифрових ресурсів у популярних інтерактивних мультимедійних фреймворках PixiJS та Phaser, та з'ясовано їх обмеження у випадку композитних ресурсів, що складаються з кількох взаємозалежних файлів. Наведено модель подання процесу завантаження цифрових ресурсів у вигляді динамічного графа завдань, що містить синхронні та асинхронні операції та відображає логічні залежності між підресурсами. Розроблено алгоритм паралельно-послідовного завантаження цифрових ресурсів з контрольованою кількістю одночасних HTTP-з'єднань, який забезпечує поєднання послідовних етапів ініціалізації та паралельних мережевих операцій без порушення коректності оброблення даних. Застосовано механізм централізованого керування рівнем паралелізму, що дає змогу уникати простою доступних мережевих з'єднань і водночас не перевищувати встановлений ліміт одночасних запитів у браузерному середовищі виконання. Проведено експериментальне порівняння запропонованого підходу зі зазначеними стратегіями на різних типах мультимедійних даних та платформах виконання з використанням багаторазових повторюваних вимірювань і статистичного відсіву аномальних значень. Унаслідок проведеного експерименту встановлено, що найбільші покращення досягаються для наборів даних із композитними ресурсами, де запропонований підхід забезпечує скорочення часу завантаження цифрових ресурсів приблизно на 30 % порівняно зі стандартним механізмом PixiJS, на 9 % порівняно з фреймворком Phaser та на 65 % порівняно з послідовною стратегією завантаження. Досліджено, що застосування динамічного графа завдань забезпечує стабільніше використання доступного рівня мережевого паралелізму та зменшує ймовірність виникнення простоїв HTTP-з'єднань. Продемонстровано придатність запропонованого підходу для використання у клієнтських мультимедійних застосунках, де критичними є часові характеристики ініціалізації, передбачуваність процесу завантаження цифрових ресурсів та ефективне використання мережевих ресурсів.

Ключові слова: завантаження медіаресурсів; мультимедійні застосунки; оптимізація продуктивності; динамічний граф завдань; паралелізм завдань.

Вступ / Introduction

У сучасних мультимедійних програмних системах важливу роль відіграє ефективне завантаження цифрових ресурсів, зокрема – зображень, анімацій, текстур і структурованих даних. У веборієнтованих та гібридних мобільних застосунках такі ресурси часто мають складну внутрішню структуру та залежності між окремими складовими, що вимагає дотримання визначеного порядку завантаження цифрових ресурсів. Водночас, середовища виконання на основі технології WebView (наприклад, Cordova або Capacitor) накладають обмеження на кількість одночасних мережевих з'єднань, що ускладнює ефективне використання ефекту паралелізму мережевих операцій.

Типові підходи до завантаження цифрових ресурсів або ігнорують залежності між складовими цих ресурсів, або використовують послідовне їх завантаження, що призводить до нераціонального використання доступних ресурсів. Наявні засоби керування ресурсами в популярних фреймворках, зокрема – PixiJS та Phaser, реалізовано у вигляді вбудованих механізмів із обмеженими можливостями явного керування плануванням підресурсів, що знижує їхню ефективність у складних сценаріях завантаження цифрових ресурсів.

З огляду на це, актуальним є розроблення підходів до організації завантаження цифрових ресурсів, які поєднували б дотримання залежностей між складовими цих ресурсів із максимально можливим паралельним виконанням з урахуванням обмежень середовища виконання.

Актуальність дослідження зумовлена потребою підвищення ефективності завантаження складних мультимедійних ресурсів у веб та мобільних ігрових застосунках за умов обмеженої кількості одночасних мережевих з'єднань. Особливої уваги потребують композит-

Білас Юрій Юрійович, аспірант, кафедра автоматизованих систем управління. **Email:** yurii.y.bilas@lpnu.ua; https://orcid.org/0009-0006-0051-5081
Різник Володимир Васильович, д-р техн. наук, професор, кафедра автоматизованих систем управління. **Email:** volodymyr.v.riznyk@lpnu.ua; https://orcid.org/0000-0002-3880-4595
Цитування за ДСТУ: Білас Ю. Ю., Різник В. В. Алгоритм паралельно-послідовного завантаження цифрових ресурсів із контрольованою кількістю одночасних завантажень. Науковий вісник НЛТУ України. 2026, т. 36, № 1. С. 144–157.
Citation APA: Bilas, Yu. Yu., & Riznyk, V. V. (2026). Algorithm for parallel-sequential loading of digital resources with control of maximum simultaneous loading operations. *Scientific Bulletin of UNFU*, 36(1), 144–157. https://doi.org/10.36930/40360116

ні ресурси, такі як анімації Spine (система скелетної 2D-анімації для ігор), що складаються з кількох взаємозалежних файлів і потребують поєднання послідовних і паралельних етапів завантаження цифрових ресурсів. Тому нагальним завданням є розроблення та реалізація методу організації процедури завантаження цифрових ресурсів із внутрішніми залежностями шляхом динамічного формування графа завдань і керування рівнем їх паралельного виконання.

Об'єкт дослідження – завантаження мультимедійних цифрових ресурсів із внутрішніми залежностями у веборієнтованих мультимедійних застосунках.

Предмет дослідження – динамічні графові моделі подання залежностей між підресурсами та алгоритми гібридного паралельно-послідовного планування їх завантаження, що дасть змогу ефективніше використовувати мережеві ресурси, і, як наслідок, пришвидшити завантаження цих ресурсів.

Мета роботи – розробити систему керування процедурою завантаження цифрових ресурсів, яка забезпечує поєднання умови дотримання залежностей між складовими цих ресурсів із ефективним використанням доступного паралелізму мережевих операцій у межах заданих обмежень, що дасть можливість проаналізувати отримані характеристики різних стратегій їхнього завантаження.

Для досягнення зазначеної мети визначено такі основні *завдання дослідження*:

- проаналізувати останні дослідження та публікації за наявними підходами до завантаження мультимедійних цифрових ресурсів із внутрішніми залежностями у веборієнтованих мультимедійних застосунках, що забезпечить обґрунтування вимог до моделі та алгоритму планування;
- розробити модель подання процедури завантаження цифрових ресурсів у вигляді динамічного графа завдань, що дасть змогу явно задавати залежності між підресурсами та розділяти синхронні та асинхронні операції;
- реалізувати механізм гібридного паралельно-послідовного виконання завдань завантаження мережевих ресурсів та опрацювання цих даних із контрольованою кількістю одночасних операцій, що дасть змогу мінімізувати простій мережевих з'єднань;
- провести експериментальне порівняння запропонованого підходу з наявними стратегіями завантаження цифрових ресурсів на різних їхніх типах, що дасть змогу оцінити ефективність запропонованого алгоритму та обґрунтувати вибір параметрів (зокрема ліміту одночасних завантажень).

Аналіз останніх досліджень та публікацій. Спробуємо виконати це завдання стосовно таких підходів до завантаження мультимедійних цифрових ресурсів: продуктивності вебзавантаження цифрових ресурсів і мережевих обмежень; оптимізації клієнтського завантаження цифрових ресурсів; планування залежних завдань та графові моделі виконання; наявних засобів керування ресурсами у сучасних фреймворках, а також їхнього критичного аналізу і виявлення невирішених проблем.

Дослідження продуктивності вебзавантаження цифрових ресурсів та мережевих обмежень. Проблему продуктивності завантаження цифрових ресурсів у вебзастосунках неодноразово розглянуто в наукових дослідженнях, в яких проаналізовано вплив мережевих протоколів і транспортних обмежень на затримки передавання даних. У роботах, що аналізують особливості HTTP/1.1, показано, що обмеження на кількість одночасних TCP-з'єднань істотно впливають на тривалість завантаження сторінок і мультимедійного контенту, особливо за умов великої кількості ресурсів [2, 11]. Дослідження також підтверджують, що неконтрольоване збільшення рівня паралелізму мережевих операцій не завжди призводить до зменшення затримок, а в окремих випадках зумовлює перевантаження клієнтської сторони та деградацію продуктивності [18]. Зокрема, у роботі [16] наведено узагальнені дані щодо кількості одночасних з'єднань, які підтримують поширені браузери (для HTTP/1.1 та HTTP/1.0), що вказує на практичну значущість врахування відповідних лімітів під час планування завантаження цифрових ресурсів. Отже, для більшості популярних браузерів ліміт становить 6 з'єднань.

Подальші роботи зосереджуються на порівняльному аналізі HTTP/1.1 та HTTP/2, зокрема – з погляду ефективності мультиплексування запитів і зменшення накладних витрат на встановлення з'єднань [10]. Водночас, зазначається, що значна кількість реальних клієнтських середовищ, зокрема – гібридних мобільних застосунків на основі WebView, продовжують використовувати механізми HTTP/1.1 з фіксованими обмеженнями на кількість паралельних з'єднань [16]. Це підтверджує, що проблема оптимального використання ефекту паралелізму мережевих операцій залишається актуальною та потребує програмних рішень на клієнтській стороні.

Методи оптимізації клієнтського завантаження цифрових ресурсів. У науковій літературі запропоновано низку підходів до оптимізації клієнтського завантаження цифрових ресурсів, зокрема – шляхом зменшення кількості мережевих запитів або агрегування кількох об'єктів в один. Так, у праці [5] розглядають методи пакування файлів, що дають змогу скоротити накладні витрати, пов'язані з ініціалізацією з'єднань. Інші дослідження пропонують алгоритми пріоритизації завантаження контенту або оптимізації черг запитів задля зменшення затримок відображення даних [1, 3, 16].

Водночас, більшість таких підходів орієнтовані на оптимізацію транспортного рівня або зменшення загальної кількості запитів і не враховують внутрішніх залежностей між складовими складних мультимедійних ресурсів. Як наслідок, вони малопридатні для сценаріїв, у яких коректність і ефективність завантаження цифрових ресурсів визначається не тільки кількістю ресурсів, а й порядком їх оброблення.

Планування залежних завдань та графові моделі виконання. У завданнях із внутрішніми залежностями між операціями широко застосовуються графові моделі подання процесів виконання, де вершини відповідають окремим завданням, а ребра – залежностям між ними. Подібні підходи використовуються у системах паралельних обчислень і планування виконання завдань, зокрема – у вигляді орієнтованих ациклічних графів DAG (англ. *Directed Acyclic Graph*) [7], що дають змогу формалізувати послідовність і можливості паралельного виконання операцій. Проте більшість відомих методів графового планування орієнтовані на серверні або обчислювальні середовища та не враховують специфіки клієнтських вебзастосунків. Зокрема, вони не беруть до уваги обмежену кількість одночасних мережевих операцій, а також необхідність динамічного формування графа завдань у процесі виконання, що є характерним для завантаження композитних мультимедійних ресурсів.

Засоби керування ресурсами в сучасних фреймворках. У сучасних ігрових та мультимедійних фреймворках, таких як PixiJS та Phaser, керування цифровими ресурсами реалізується за допомогою вбудованих механізмів завантаження цифрових ресурсів, призначених для спрощення процесу розроблення застосунків [13, 12]. Вихідний код відповідних завантажувачів є відкритим і доступним для аналізу, проте, їхні моделі планування завантаження цифрових ресурсів орієнтовані переважно на базові сценарії роботи з наборами файлів.

У фреймворку PixiJS завантаження цифрових ресурсів здійснюється шляхом формування набору асинхронних операцій, які виконуються паралельно без явного обмеження кількості одночасних мережевих з'єднань [13]. Підресурси складних ресурсів, зокрема – анімацій, завантажуються всередині відповідних парсерів, що зумовлює локальне керування порядком і паралельністю без централізованого механізму глобального планування. Такий підхід ускладнює контроль за загальним рівнем паралелізму мережевих операцій та не дає змоги формалізувати залежності між підресурсами у вигляді єдиної моделі виконання.

На відміну від цього, у фреймворку Phaser реалізовано механізм обмеження кількості одночасних завантажень, який дає змогу задавати максимальну кількість паралельних мережевих операцій [12]. Завантажувач підтримує пул активних завантажень із фіксованою верхньою межею та динамічно поповнює його в міру завершення окремих операцій. Водночас, планування завантаження цифрових ресурсів у фреймворку Phaser здійснюється на рівні окремих файлів і не враховує структурних залежностей між складовими композитних ресурсів. Окрім цього, кожен паралельний слот у пулі займається зазвичай не тільки асинхронною роботою, яка вимагає доступу до обмеженої кількості мережевих операцій, а ще й синхронною роботою, в такий спосіб займаючи даремно пул.

Критичний аналіз і виявлення невирішених проблем. Отже, аналіз відомих досліджень свідчить, що більшість наявних робіт зосереджені переважно на оптимізації транспортного рівня або зменшенні кількості мережевих запитів. Водночас, питання динамічного планування завантаження взаємозалежних мультимедійних ресурсів із контрольованим рівнем паралелізму мережевих операцій у клієнтських мультимедійних застосунках залишається недостатньо дослідженим. Це зумовлює доцільність розроблення спеціалізованих засобів керування процесом завантаження цифрових ресурсів, що і є предметом дослідження цієї роботи.

Матеріали та методи дослідження. Використано метод аналізу та узагальнення для вивчення підходів до організації процесу завантаження цифрових ресурсів у клієнтських вебзастосунках. Дослідження базується на використанні графових моделей подання процесів із внутрішніми залежностями та обмеженнями на кількість одночасних HTTP-з'єднань. Оцінювання ефективності запропонованого підходу проведено шляхом експериментального порівняння різних стратегій завантаження ресурсів за фіксованого ліміту паралельних мережевих з'єднань у клієнтському середовищі виконання.

Результати дослідження та їх обговорення / Research results and their discussion

Методика проведення експерименту. Для досягнення поставленої мети роботи та виконання сформульованих завдань дослідження розроблено відповідне програмне забезпечення для експериментального порівняння різних стратегій завантаження цифрових ресурсів.

Програмне забезпечення розроблене мовою програмування TypeScript, причина вибору якої полягає в її статичній типізації, що зменшує кількість помилок та покращує "читабельність" програмного коду. Також було обрано технологію Capacitor для створення нативного додатку для мобільної ОС Android та отримання доступу до WebView, який використовується у великій кількості мобільних додатків (зокрема мультимедійних та ігрових). Заготовку у вигляді Capacitor-проєкту, який однією командою термінала встановлюється на Android пристрій, згенеровано з допомогою ШІ – використовувався редактор Cursor (стандарт індустрії) з агентом із автоматичною моделлю ШІ.

Розроблене ПЗ реалізовує або застосовує вже готові алгоритми для завантаження цифрових ресурсів через протокол HTTP 1.0. У табл. 1 наведено перелік загальної тривалості пересилань повідомлень та короткий опис кожного. На рис. 1 наведено блок-схему алгоритму завантаження цифрових ресурсів "З обмеженням".

На цьому рисунку на вхід подаються items – ресурси, які потрібно опрацювати, count – кількість ресурсів, limit – максимальна кількість одночасних завантажень. Алгоритм використовує RUN – опрацювання конкретного ресурсу, WAIT_ANY – очікування, поки хоча б одна з операцій завершиться, WAIT_ALL – очікування, поки всі операції завершаться.

Табл. 1. Порівняння загальної тривалості пересилання повідомлення (з кодуванням, мутацією та декодуванням) / Comparison of the total time for sending a message (including encoding, mutation, and decoding)

Назва українською	Кодова назва англійською	Короткий опис
Один за одним	one_by_one	Завантаження наступного ресурсу починається тільки після завершення попереднього.
Усі водночас	all_at_once	Завантаження всіх ресурсів починається водночас.
З обмеженням	limited	Водночас відбувається завантаження максимум N ресурсів.
PixiJS за замовчуванням	pixi_default	Стандартний алгоритм завантаження цифрових ресурсів, вбудований у фреймворк.
Phaser за замовчуванням	phaser_default	Стандартний алгоритм завантаження цифрових ресурсів, вбудований у фреймворк.
Алгоритм ППЗ-ККЗ (**П**аралельно-**П**ослідовний алгоритм **З**авантаження з **К**онтрольованою **К**ількістю одночасних **З**авантажень)	plus	Водночас відбувається завантаження максимум шести ресурсів, а процес завантаження кожного з ресурсів наведено, як підграф з синхронних та асинхронних операцій. Усі підграфи поєднані в загальний граф, що опрацьовується централізовано. Максимально уникається простій HTTP з'єднань.

Алгоритм PixiJS працює схоже до "Усі водночас". Усі ресурси, які є в черзі на завантаження, водночас, відправляються на опрацювання. Жодного контролю максимального паралелізму мережевих операцій немає. Також немає централізованого контролю: в PixiJS є концепт окремих сутностей-завантажувачів (Loader), кожен з яких займається своїми ресурсами. З боку користувача фреймворка це є інтуїтивно зрозуміло, і зручно. Проте, з боку завантаження цифрових ресурсів є проблеми такі ж, як і в методу "Усі водночас", і про них буде в розділах нижче.

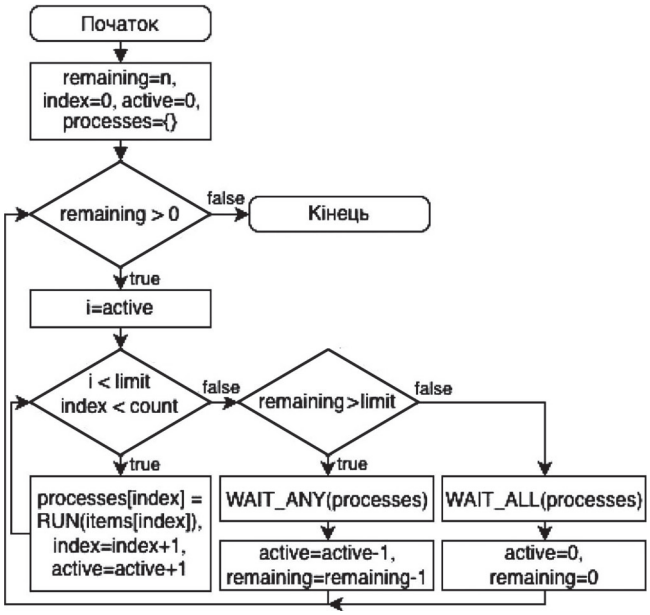

Рис. 1. Схема алгоритму завантаження цифрових ресурсів "З обмеженням" / Algorithm diagram for loading digital resources "With restrictions"

Алгоритм Phaser, на відміну від PixiJS, працює схоже до алгоритму "З обмеженням". Фреймворк Phaser встановлює ліміт файлів, що завантажуються водночас, і цей ліміт – 6 на Android та 32 на інших системах. Проте у фреймворку Phaser завантаження кожного з файлів є операціями, що містять як асинхронні завантаження цифрових ресурсів з допомогою HTTP, так і синхронні, подекуди тривалі дії (наприклад парсинг файлів Spine: атласу з текстурами та файлу скелетної анімації). Такий підхід є близьким до оптимального, проте, відсутність розділення завантаження цифрових ресурсів на синхронну та асинхронну складові призводить до простою HTTP з'єднань. Варто також зазначити, що:

- завантаження цифрових ресурсів відбувається централізовано;
- в Phaser відсутня концепція зручного завантажувача сутності;
- також наразі цей фреймворк не підтримує роботу з пріоритетизацією завдань, а також не дає змоги утворювати складні логічні зв'язки між файлами та динамічно припиняти їх завантаження.

Алгоритм ППЗ-ККЗ (**П**аралельно-**П**ослідовний алгоритм **З**авантаження з **К**онтрольованою **К**ількістю одночасних **З**авантажень) націлений на вирішення всіх проблем, згаданих вище. Цей алгоритм аналогічно, як і алгоритм "З обмеженням" та стандартний алгоритм фреймворку Phaser, встановлює обмеження у вигляді максимально шести (для всіх ОС) одночасних завантажень з допомогою HTTP. Процес завантаження цифрових ресурсів подається у вигляді динамічного графа завдань, окремі композитні цифрові ресурси якого представлені відповідними підграфами. Листкові вершини графа відповідають елементарним операціям: синхронні операції та асинхронні завантаження цифрових ресурсів з допомогою HTTP; а внутрішні вузли є композитами двох видів: послідовності (англ. *Sequence*) та групи паралельних (англ. *Parallel*) завдань. Перші можуть виконуватись тільки один за одним, а другі – можуть виконуватись водночас, незалежно одне від одного. Такі підграфи кожного з ресурсів об'єднуються в більший граф централізованої системи керування процедурою завантаження цифрових ресурсів, де в корені є головний вузол паралельного типу.

Також цей графовий підхід чудово працює з композитуванням ресурсів: підграф, що стосується завантаження одного зі складних ресурсів, може містити ще один вкладений підграф, що стосується залежного ресурсу. Графова структура дає змогу утворювати також складні зв'язки (наприклад, горизонтальна залежність одного файлу від завантаження іншого файлу), проте, це окрема дещо складніша тема, яка виходить за межі даної роботи й описуватиметься в наступному дослідженні. На рис. 2 наведено приклад графа завдань завантаження цифрових ресурсів, з яким працює алгоритм, а на рис. 3 – блок-схему алгоритму ППЗ-ККЗ. На цій схемі зображено програмну логіку, яка повинна виконуватись щоразу, коли відбувається додавання нових завдань у граф, чи завершується яке-небудь завдання. Після завершення, завдання також видаляється зі списку async_tasks або sync_task занулюється.

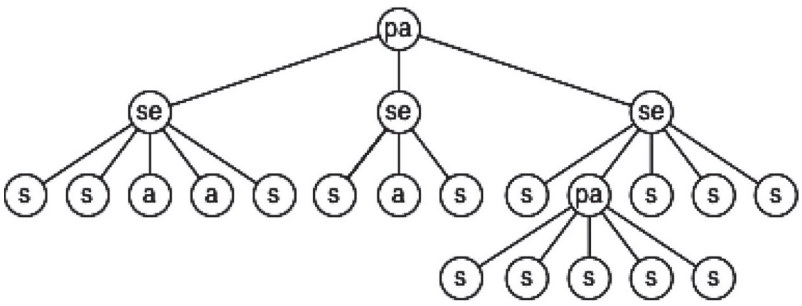

Рис. 2. Приклад графа завдань завантаження цифрових ресурсів / Example of a digital resources loading task graph

Для проведення експерименту використовуються різні набори ресурсів. У табл. 2 наведено їх перелік та короткі описи.

Найбільша увага до набору "Різноманітні ресурси", адже він найкраще представляє ситуацію, що трапляється в реальних сучасних мультимедійних додатках, зокрема – індустрії мобільних ігор. Цей набір містить тільки малі зображення (10 шт.) та 2 групи анімацій зі сумарним розміром кожної до 1 МБ, тобто 16 файлів загалом.

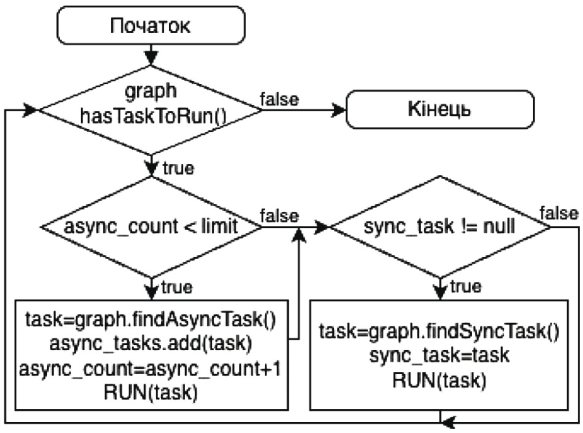

Рис. 3. Блок-схема алгоритму ППЗ-ККЗ / Flowchart of the PSL-CSL algorithm

Під час експерименту файли беруться завжди випадково з кожного набору, а також перемішуються у випадковому порядку. Після кожної ітерації експерименту проводиться звільнення завантажених цифрових ресурсів. Окрім цього, між кожною ітерацією експерименту передбачається додаткова пауза 100 мс, щоб стабілізувати стан середовища виконання, що давало змогу мінімізувати вплив побічних факторів, пов'язаних із кешуванням та фоновою активністю.

Результати всіх ітерацій експерименту зберігаються у списку для подальшого статистичного опрацювання. Зібрані дані очищаються від викидів (англ. outlier) з допомогою стандартної оцінки (або z-оцінки, англ. z-score; форм), відкидаючи показники, оцінка яких за модулем є більшою за 1.5. Для очищених даних підраховується середнє арифметичне значення та медіана.

Табл. 2. Набори цифрових ресурсів, використані в експерименті / Digital resource sets used in the experiment

Назва українською	Кодова назва англійською	Короткий опис
Великі зображення	big_images	Розмір від 1.8 МБ та більше. 10 файлів.
Середні зображення	medium_images	Розмір від 100 КБ до 600 КБ. 10 файлів.
Малі зображення	small_images	Розмір від 5 КБ до 60 КБ. 10 Файлів.
Різноманітні зображення	diverse_images	Поєднання малих, середніх та великих файлів. 30 файлів.
Spine анімації	spine	Комбінації файлів анімацій Spine зі сумарними розмірами від 150 КБ до 2 МБ. 3 групи файлів, загалом 11 одиниць.
Різноманітні ресурси	misc	Комбінація файлів анімацій Spine та зображень.

Опис інтерфейсу розробленого програмного забезпечення. Для досягнення поставленої мети та виконання сформульованих завдань розроблено програмне забезпечення для збирання та аналізу результатів експериментів із завантаженням ресурсів. На рис. 4 зображено вигляд програми на початку її роботи. Програма реалізована без графічного користувацького інтерфейсу, а натомість використовує командний рядок браузера, адже її ціль – це тільки автоматизоване проведення експериментів та отримання статистики. На рис. 4 наведено приклад виконання двох експериментів з різними кількостями ітерацій, типами ресурсів та стратегіями завантаження цифрових ресурсів, а також видно проміжні результати кожної ітерації та остаточні результати: медіана та середнє арифметичне.

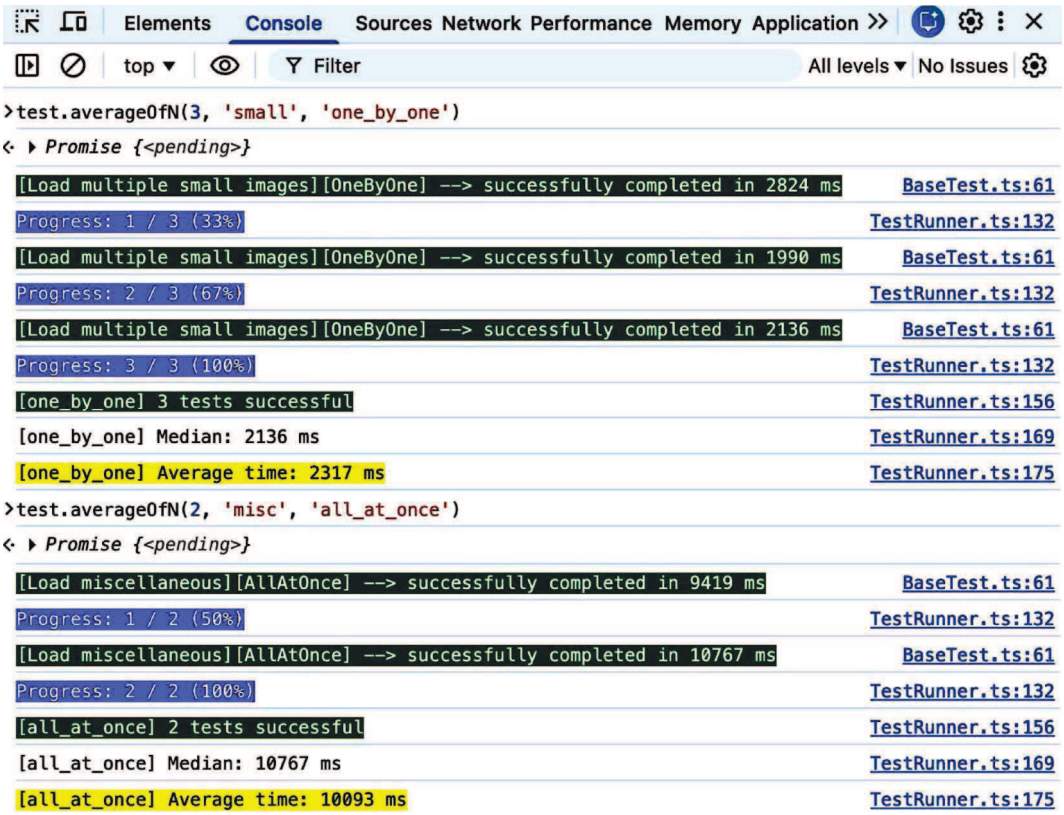

Рис. 4. Вигляд програмного забезпечення для проведення експериментів із завантаженням цифрових ресурсів / A view of the software for conducting digital resource loading experiments

Програмне забезпечення є простим, зручним у використанні (зважаючи на його призначення та цільову аудиторію), а його інтерфейс є лаконічним і інтуїтивним. З погляду продуктивності, програмний застосунок не має жодних візуальних зависань, швидко відгукується на дії користувача.

Архітектура програмного застосунку. Проаналізуємо діаграму UML-класів для розробленого програмного застосунку (рис. 5) для кращого розуміння, як він влаштований всередині. З діаграми зрозуміло, що головною точкою входу є сутність TestRunner, яка відповідає за те, щоб запускати серії тестів та збирати результати. Статистичне опрацювання результатів відокремлене в клас TestStatistics. TestRunner працює з окремою ітерацією експерименту, яка представлена класом Test. Кожен тест завантаження цифрових ресурсів визначається тим, які файли завантажувались та якою стратегією. З діаграми видно, що до цих трьох видів сутностей застосовано принцип "композиція, а не наслідування" (англ. *Composition over inheritance*), що дало змогу уникнути надмірної кількості класів. Видно також, що деякі генератори файлів мають залежності від інших сутностей цього типу. У схемі наведено шість стратегій завантаження цифрових ресурсів, дві з яких є простими стандартними підходами з популярних фреймворків PixiJS та Phaser. Решта чотири стратегії завантаження цифрових ресурсів наслідують спільний клас BaseCustomLoaderStrategy. Серед них:

- LoaderStrategyOneByOne – "один за одним";
- LoaderStrategyAllAtOnce – "усі водночас";
- LoaderStrategyLimited – "з обмеженням";
- LoaderStrategyPlus – алгоритм ППЗ-ККЗ.

На рис. 5 зображено не всю архітектуру програмного застосунку, а тільки загальний огляд її верхнього рівня. На рис. 6 наведено деталізацію архітектури ПЗ на нижчому рівні, який стосується алгоритму ППЗ-ККЗ, тобто розкриває глибші зв'язки сутності LoaderStrategyPlus. Для зручності, абстракції зображено світлими прямокутниками, а конкретні реалізації – сірим.

З діаграм (див. рис. 5, 6) видно, що архітектура модуля ППЗ-ККЗ побудована з урахуванням принципів SOLID [8]: відповідальність класів розділена (SRP); нові типи ресурсів та стратегій їх завантаження додаються без зміни наявного коду (OCP); абстракції дають можливість коректно підставляти реалізації (LSP); інтерфейси вузькі за призначенням (ISP); залежності спрямовані на абстракції, а не на конкретні класи (DIP), за винятком точки входу LoaderStrategyPlus, де допустимо створення конкретних реалізацій.

З діаграми можна простежити ланцюжок залежностей. Стратегія LoaderStrategyPlus взаємодіє з конкретною реалізацією завантажувача LoaderPlus та централізованої системи управління завантаженням LoadSystem. LoaderPlus оформлює запити на завантаження цифрових ресурсів у сутності типу IAssetData. Логіка системи завантаження цифрових ресурсів розділена на окремі підмодулі, а саме: LoadCoordinator, TasksFactory та LoadingGraph.

LoadCoordinator відповідає за те, щоб запускати на виконання певну кількість завдань (Task) певного типу. Для мережевих операцій завантаження цифрових ресурсів встановлений ліміт на максимум шість операцій водночас, виходячи з описаних в наступному розділі експериментів, а також спираючись на дослідження інших авторів [16]. Ця сутність завдань також чудово підходить для задавання обмеження максимальної кількості синхронних блокувальних операцій, що не використовують мережу. Це може бути потрібним тоді, коли, у разі відсутності таких лімітів, кількість блокувальних операцій може бути надмірно великою на один кадр промальовки програми, що може спричиняти видимі зависання її роботи, які недопустимі в деяких випадках, найчастіше в ігрових застосунках (або інших мультимедійних).

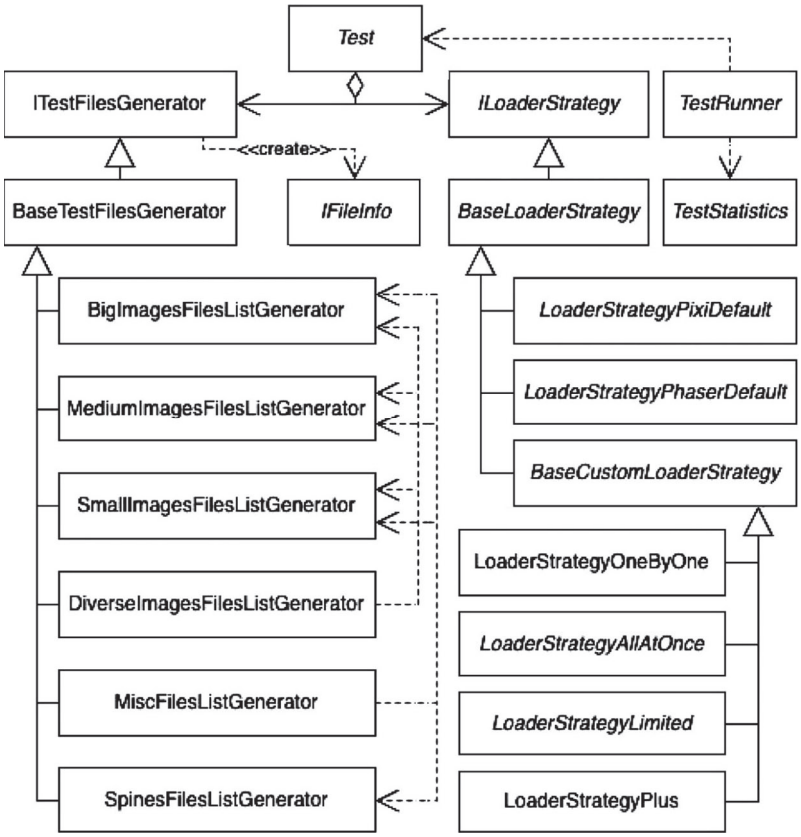

Рис. 5. Діаграма UML-класів для загальної архітектури програмного застосунку / UML class diagram for the overall software application

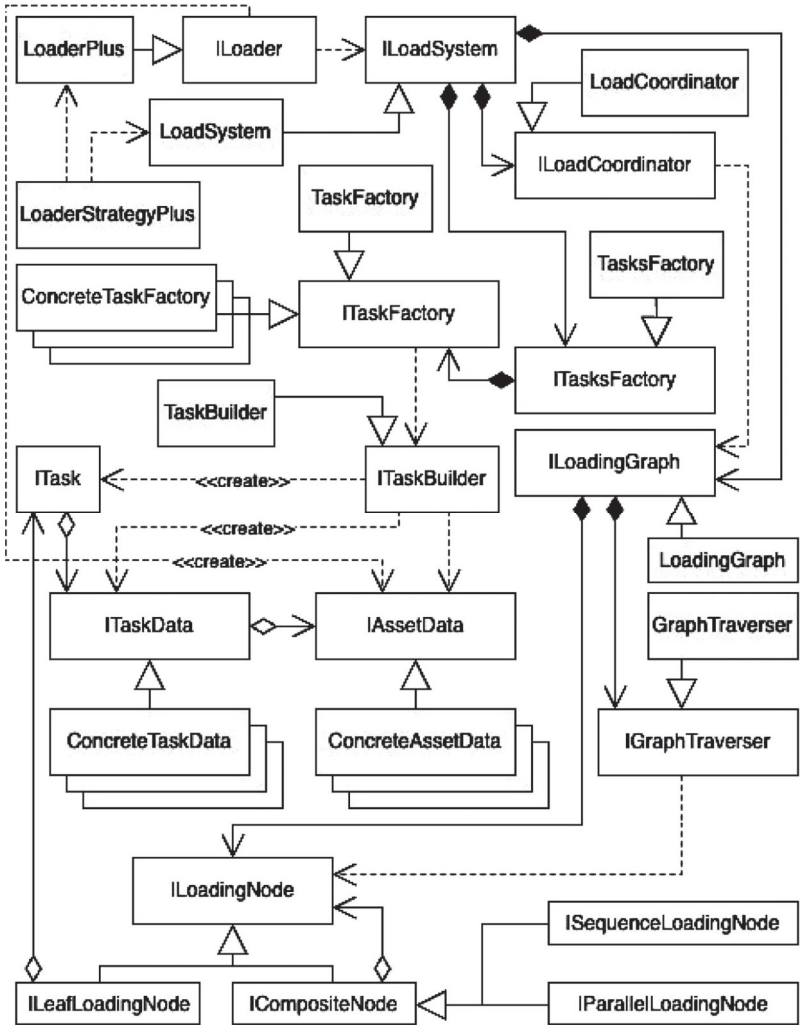

Рис. 6. Діаграма UML-класів для сегмента архітектури програмного застосунку, пов'язаного зі стратегією LoaderStrategyPlus (ППЗ-ККЗ) / UML class diagram for the software's architecture segment related to LoaderStrategyPlus (PSL-CSL, Parallel-Sequential Loading with Control of maximum Simultaneous Loading operations)

Алгоритм ППЗ-ККЗ практично застосував один з авторів цієї роботи в реальному комерційному ПЗ компанії, назву якої не можемо вказати через угоду про нерозголошення. Для вирішення згаданої проблеми надмірної кількості синхронних операцій завантаження цифрових ресурсів й було вирішено застосувати певні обмеження, які визначались показником FPS (англ. *Frames Per Second*). Також саме LoadCoordinator є найкращим місцем для впровадження різних підстратегій завантаження цифрових ресурсів. Наприклад, у згаданій компанії експериментально використовувались підстратегії Idle (для простою, коли немає ресурсів у черзі), Urgent (коли є ресурси, які потрібно завантажити терміново) та RespectFPS (коли є ресурси в черзі, проте, вони не потрібні одразу ж, а можуть бути завантажені поступово пізніше).

TasksFactory – це фасад, який надає доступ до різноманітних сутностей-фабрик (на схемі позначено, як ConcreteTaskFactory), серед яких TextureTaskFactory, SpineTaskFactory тощо. Дані фабрики використовують зручні сутності TaskBuilder для створення підграфів завантаження цифрових ресурсів. TaskBuilder, на основі створених перед тим Loader-ом запитів на завантаження IAssetData, утворює набір завдань типу ITask. Також TaskBuilder створює для цього набору завдань спільний об'єкт типу ITaskData (його можна сприймати, як контекст виконання складної процедури завантаження одного ресурсу). Усі завдання ITask зчитують стан завантаження цифрових ресурсів з цього спільного об'єкта, а також записують у нього свої проміжні результати. Контекст ITaskData, звісно ж, має посилання на відповідний запит IAssetData.

LoadingGraph – це сутність, що представляє граф завдань та їх композитів. Цей граф містить підграфи окремих ресурсів, а ті, водночас, теж можуть містити інші підграфи. Використовується шаблон проєктування "Композит". ILeafLoadingNode – листок графа – є тією складовою графа, яку можна виконувати. Саме тому вони мають посилання на завдання ITask. Завдання діляться на звичайні синхронні та мережеві асинхронні. Композити в цьому графі бувають двох видів: послідовні (ISequenceLoadingNode) та паралельні (IParallelLoadingNode). Перші працюють так, що наступне завдання послідовності не можна виконувати, поки не виконається повністю попереднє; другі ж не накладають обмежень на порядок виконання, і дають можливість виконання групи завдань водночас (паралельно).

Граф також має допоміжну сутність GraphTraverser, завданням якої є пошук наступного листка, який можна виконати. Цей клас може бути різної складності. У даній роботі реалізовано найпростіший алгоритм підбору наступного листка (рис. 7). У цьому рисунку на вхід передається кореневий вузол root та булеве значення sync, яке визначає тип завдання, яке потрібно повернути.

Протягом пошуку потрібного типу завдання алгоритм кешує знайдені результати іншого типу. Пізніше ці результати можна перевикористати, щоб виконувати пошук швидше. Натомість в комерційному продукті згаданої вище компанії використовується набагато складніший підхід, який дає змогу задовольняти потреби різних згаданих вище підстратегій завантаження цифрових ресурсів, а також містять результати опрацювання пріоритизації (тобто, можливості обирання до завантаження цифрових ресурсів, враховуючи їх відносний пріоритет).

Водночас, на цій схемі: s – синхронне завдання, a – асинхронне (мережеве) завдання, se – послідовність, pa – група завдань, що можуть виконуватись паралельно.

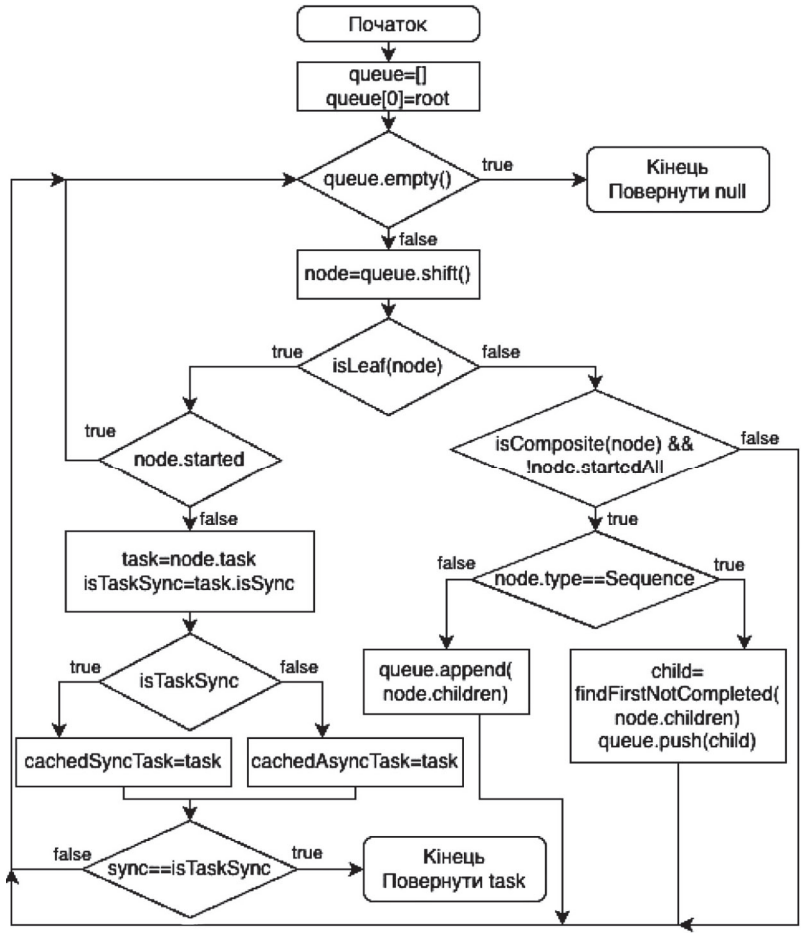

Рис. 7. Блок-схема алгоритму пошуку наступного виконуваного листка у графі завдань (GraphTraverser) / Flowchart of the algorithm for finding the next executable leaf in the task graph (GraphTraverser)

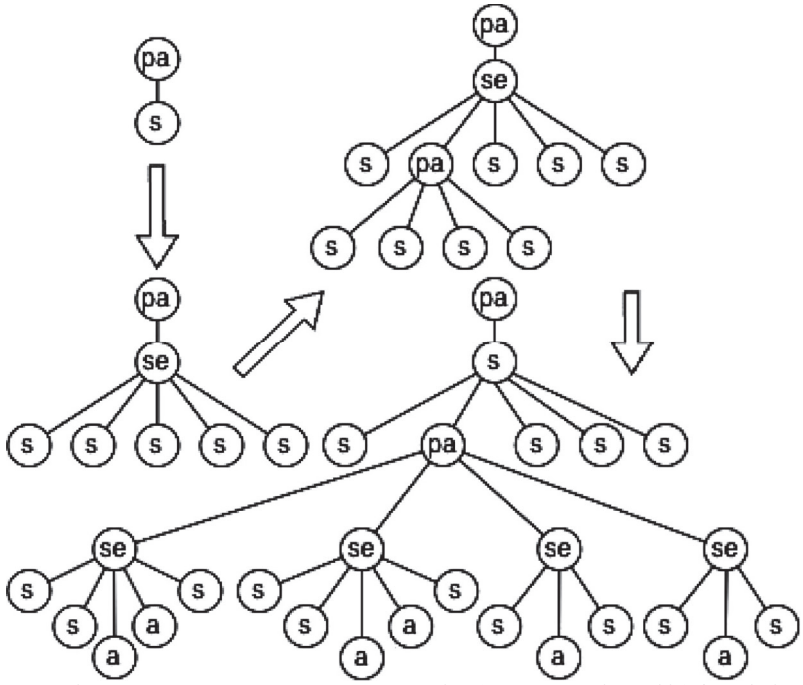

Рис. 8. Еволюція графа завдань під час завантаження ресурсу типу Spine (розкриття підграфів) / Evolution of the task graph during loading of a Spine-type resource (subgraph expansion)

Для підтримки коректності реалізації алгоритму ППЗ-ККЗ та сумісних компонентів (граф завдань, фабрики завдань, координація завантаження цифрових ресурсів) у модулі тестів розроблено 19 unit-тестів, які покривають основні сценарії формування графа, вибору наступного завдання та інтеграції з різними типами ресурсів.

Результати проведеного експерименту. Використовуючи розроблену систему, проведено серію експериментів зі завантаження цифрових ресурсів різними підходами (табл. 3-8). А також продемонстровано, як на тривалість завантаження цифрових ресурсів впливає ліміт їхньої кількості, що накладається на максимальну кількість одночасних завантажень ресурсів в алгоритмі ППЗ-ККЗ (табл. 9).

Табл. 3. Результати тестів різних стратегій завантаження цифрових ресурсів для набору даних "big_images" / Test results of different strategies of digital resources loading for the "medium_images" dataset

Стратегія	Медіана, мс	Середнє, мс	Відносне, %
one_by_one	177386	176132	+182
all_at_once	9536	64391	+3
pixi_default	63617	63391	+1
limited	61760	62760	+0
phaser_default	62002	61443	-2
plus	62507	62489	+0

Примітка: для кожної стратегії виконано 10 ітерацій; дані посортовано за середнім; відносне значення тривалості завантаження цифрових ресурсів розраховане порівняно зі стратегією "plus".

Табл. 4. Результати тестів різних стратегій завантаження цифрових ресурсів для набору даних "medium_images" / Test results of different strategies of digital resources loading for the "medium_images" dataset

Стратегія	Медіана, мс	Середнє, мс	Відносне, %
one_by_one	15084	14953	+102
all_at_once	9536	9424	+27
pixi_default	8898	8969	+21
limited	7768	7675	+4
phaser_default	7441	7385	+0
plus	7700	7404	+0

Примітка: для кожної стратегії виконано 100 ітерацій; дані посортовано за середнім; відносне значення тривалості завантаження цифрових ресурсів розраховане порівняно зі стратегією "plus".

Табл. 5. Результати тестів різних стратегій завантаження цифрових ресурсів для набору даних "small_images" / Test results of different strategies of digital resources loading for the "small_images" dataset

Стратегія	Медіана, мс	Середнє, мс	Відносне, %
one_by_one	3390	3367	+288
all_at_once	1050	967	+12
pixi_default	952	954	+10
limited	921	949	+9
phaser_default	971	935	+8
plus	868	867	+0

Примітка: для кожної стратегії виконано 500 ітерацій; дані посортовано за середнім; відносне значення тривалості завантаження цифрових ресурсів розраховане порівняно зі стратегією "plus".

З отриманих даних (див. табл. 3-8) бачимо, що стратегія завантаження цифрових ресурсів "Один за одним", будучи найпримітивнішою, справляється найгірше з усіх. З великим відривом від неї, майже на всіх других місцях за тривалістю завантаження цифрових ресурсів, є ще одна дуже проста стратегія завантаження "Усі водночас". Для дуже великих наборів даних (див. табл. 3 та 6) різниці в результатах є в зоні погрішності. Краще помітна різниця між стратегіями завантаження, коли файли менші, і їх багато (див. табл. 5 та 8).

Табл. 6. Результати тестів різних стратегій завантаження цифрових ресурсів для набору даних "diverse_images" / Test results of different strategies of digital resources loading for the "diverse_images" dataset

Стратегія	Медіана, мс	Середнє, мс	Відносне, %
one_by_one	198475	210965	+170
all_at_once	80054	79831	+2
pixi_default	78886	79098	+1
limited	79001	78817	+1
phaser_default	76450	76408	-2
plus	79271	78037	+0

Примітка: для кожної стратегії виконано 10 ітерацій; дані посортовано за середнім; відносне значення тривалості завантаження цифрових ресурсів розраховане порівняно зі стратегією "plus".

Табл. 7. Результати тестів різних стратегій завантаження цифрових ресурсів для набору даних "spine" / Test results of different strategies of digital resources loading for the "spine" dataset

Стратегія	Медіана, мс	Середнє, мс	Відносне, %
one_by_one	20269	19838	+153
all_at_once	10108	9652	+23
pixi_default	9792	9461	+21
limited	9024	8923	+14
phaser_default	8611	8545	+9
plus	8251	7849	+0

Примітка: для кожної стратегії виконано 50 ітерацій; дані посортовано за середнім; відносне значення тривалості завантаження цифрових ресурсів розраховане порівняно зі стратегією "plus".

Табл. 8. Результати тестів різних стратегій завантаження цифрових ресурсів для набору даних "misc" / Test results of different strategies of digital resources loading for the "misc" dataset

Стратегія	Медіана, мс	Середнє, мс	Відносне, %
one_by_one	12248	12261	+188
all_at_once	7071	7164	+68
pixi_default	6414	6086	+43
limited	5127	4955	+16
phaser_default	4734	4670	+10
plus	4032	4256	+0

Примітка: для кожної стратегії виконано 200 ітерацій; дані посортовано за середнім; відносне значення тривалості завантаження цифрових ресурсів розраховане порівняно зі стратегією "plus".

Табл. 9. Результати тестів різних значень ліміту максимальних завантажень цифрових ресурсів для стратегії "plus" на наборі даних "misc" / Test results for different values of the maximum simultaneous digital resources loading limit for the "plus" strategy on the "misc" dataset

Ліміт	Медіана, мс	Середнє, мс
4	4716	4740
5	4581	4612
6	4196	4349
7	4612	4544
8	4467	4645

Примітка: для кожної стратегії виконано 100 ітерацій.

Помітно, що стандартна стратегія завантаження PixiJS, як і передбачалось, є найчастіше схожою за результатами на "Усі водночас", а алгоритм Phaser – найбільш схожий на "З обмеженням". Цікаво, що, коли

серед файлів опиняються анімації Spine (табл. 7-8), то стандартні стратегії роботи фреймворків помітно вигравають порівняно з примітивнішими варіантами. Беззаперечним лідером серед усіх випробуваних стратегій завантаження цифрових ресурсів є алгоритм ППЗ-ККЗ, який зазвичай сильно випереджав конкурентів за швидкістю, або хоча б залишався на рівні з ними (у випадку великих файлів). Найкраще алгоритм ППЗ-ККЗ проявляє себе у ситуаціях, коли потрібно завантажувати багато невеликих файлів, які часто об'єднані між собою логічно в комплексні структури (як наприклад, Spine), та які мають дороговартісні синхронні операції посеред завантаження цифрових ресурсів (як парсинг атлас-файлів та скелетної анімації Spine) (див. табл. 8). Отже, алгоритм ППЗ-ККЗ є швидшим, ніж послідовне завантаження на 65 % (майже втричі), на 30 % швидшим, ніж алгоритм, що використовується фреймворком PixiJS та на 9 % швидшим, ніж Phaser.

З даних табл. 9 бачимо, що доведена наявність оптимального значення ліміту (6 у досліджуваному середовищі), що узгоджується з емпіричними обмеженнями браузерів і підтверджує доцільність явного керування ефектом паралелізму мережевих операцій. Перевищення цього значення не призводить до подальшого покращення тривалості завантаження цифрових ресурсів, а натомість може спричинити деградацію продуктивності.

Обговорення результатів дослідження. У дослідженнях, в яких висвітлено вплив стратегій планування мережевих потоків на продуктивність вебзавантаження, показано, що вибір між послідовним і паралельним плануванням істотно впливає на кінцеві часові показники. Зокрема, в роботі з аналізу HTTP/3 і QUIC [15] продемонстровано, що агресивний паралелізм не завжди є оптимальним, а ефективність досягається за умови керованого поєднання паралельних і послідовних операцій з урахуванням характеристик середовища та пріоритетів ресурсів.

Робота [6], з оптимізації часу завантаження вебсторінок, демонструє, що структура залежностей між ресурсами істотно впливає на критичний шлях, тобто найдовшу послідовність взаємозалежних операцій, виконання яких неможливо розпаралелити, і, відповідно, на час повної готовності сторінки. Запропонований у цій роботі підхід використовує інформацію про залежності та розміри об'єктів для зміни порядку завантаження, що дає змогу зменшити загальну тривалість очікування. Концептуально цей підхід близький до запропонованої в цій роботі моделі динамічного графа завдань, у якій залежності між підресурсами задають допустимий порядок виконання та визначають можливості для паралелізації.

Існують наукові дослідження, що аналізують традиційні практики оптимізації вебпродуктивності за умов обмежень HTTP/1.x, серед яких "шардінг" (англ. domain sharding, укр. сегментування доменів) розглядають як спосіб збільшення загального рівня паралелізму шляхом розподілу ресурсів по різних доменах. У роботі [9] автори оцінюють ефективність цього підходу разом із іншими відомими техніками і показують, що "шардінг" в HTTP/1.1 може ще давати переваги, яких не досягають прості оптимізації в нових протоколах. Проте, в роботі [14] зазначено, що практика "шардінгу" має недоліки, зокрема – через те, що кожне додаткове з'єднання супроводжується накладними витратами, пов'язаними з установленням TCP- і TLS-з'єднань та фазою повільного старту.

У сучасних дослідженнях [20] з розподілених і багатозадачних систем динамічні графи завдань використовують як універсальний механізм організації асинхронного та паралельного виконання з урахуванням логічних залежностей і ресурсних обмежень. Такі моделі дають можливість розкладати складні процеси на підзадачі, які можуть виконуватися паралельно за умови збереження коректності. Хоча відповідні роботи розглядають інші предметні області (зокрема ШІ), їх методологічна основа безпосередньо застосовна до задачі завантаження мультимедійних ресурсів. У даній статті цей підхід адаптовано до специфіки роботи клієнтських вебзастосунків із жорстким обмеженням кількості одночасних HTTP-з'єднань.

Безпосередньо до проблематики цієї роботи наближається дослідження [17], у якому розглянуто задачу оптимізації порядку HTTP-запитів за фіксованого ліміту паралельних з'єднань на стороні браузера. Автори показують, що за перевищення такого ліміту частина запитів блокується та очікує в черзі, а порядок їх подальшого обслуговування істотно впливає на загальний час завантаження і час початку рендерингу. Підхід, який вони пропонують, трактує HTTP-запити як незалежні задачі планування. На відміну від цього, у даній роботі планування здійснюється на основі динамічного графа залежностей, що дає змогу враховувати композитну природу мультимедійних ресурсів і поєднувати послідовні та паралельні етапи завантаження. Отже, результати попередніх досліджень підтверджують доцільність керування порядком HTTP-запитів і слугують підґрунтям для розвитку залежнісно-орієнтованих графових моделей завантаження ресурсів.

Отже, обговорення результатів дослідження підтверджує доцільність проведення саме цього дослідження, оскільки в аналізованій літературі практично відсутні комбіновані підходи до планування завантаження взаємозалежних клієнтських ресурсів із явним графом завдань та явним обмеженням виконання одночасних мережевих операцій. Отримані результати заповнюють цю прикладну прогалину, демонструючи перевагу алгоритму ППЗ-ККЗ для композитних ресурсів та наборів із великою кількістю малих файлів.

Отже, внаслідок виконаної роботи можна сформулювати такі наукову новизну та практичну значущість результатів дослідження.

Наукова новизна отриманих результатів дослідження – отримав подальший розвиток метод організації завантаження цифрових ресурсів, який, на відміну від наявних, враховує внутрішні залежності між підресурсами на основі динамічного графа завдань із контрольованим рівнем паралелізму виконання мережевих операцій, що дало можливість поєднати паралельне виконання незалежних асинхронних мережевих операцій із послідовним виконанням синхронних етапів оброблення даних, забезпечуючи ефективніше використання обмеженої кількості HTTP-з'єднань у клієнтських середовищах виконання.

Практична значущість результатів дослідження – можна використати у веборієнтованих та гібридних мобільних мультимедійних застосунках (зокрема – на базі WebView) для скорочення тривалості завантаження

композитних ресурсів за умов обмеженої кількості одночасних HTTP-з'єднань.

Висновки / Conclusions

Розроблено систему керування процедурою завантаження цифрових ресурсів, яка забезпечує поєднання умови дотримання залежностей між складовими цих ресурсів із ефективним використанням доступного паралелізму мережевих операцій у межах заданих обмежень, що дасть можливість проаналізувати отримані характеристики різних стратегій їхнього завантаження. За результатами проведеного дослідження можна зробити такі основні висновки.

1. Розроблено модель подання процедури завантаження цифрових ресурсів у вигляді динамічного графа завдань із синхронними та асинхронними операціями та композитними вузлами (послідовність, паралельна група), що дало змогу формалізувати внутрішні залежності між підресурсами та визначити допустимі порядки їх виконання з урахуванням можливостей паралелізму.
2. Реалізовано алгоритм ППЗ-ККЗ з централізованим плануванням та обмеженням виконання одночасних мережевих операцій (наприклад, 6), що дало змогу мінімізувати простій з'єднань за наявності синхронних етапів.
3. Проведено експериментальне порівняння зі стратегіями завантаження цифрових ресурсів "один за одним", "усі водночас", "з обмеженням" та вбудованими механізмами PixiJS та Phaser на наборах великих, середніх та малих зображень, різноманітних зображень, анімацій Spine та комбінованих ресурсів, що дало можливість оцінити вплив внутрішніх залежностей і обмежень паралелізму на загальну тривалість завантаження.
4. Встановлено, що алгоритм ППЗ-ККЗ забезпечує найкращий результат роботи у ситуаціях, коли потрібно завантажувати багато невеликих файлів, або порівняну тривалість завантаження цифрових ресурсів, особливо для композитних ресурсів із дороговартісними синхронними операціями та за обмеження паралелізму мережевих операцій, що забезпечило дещо ефективніше використання обмеженої кількості HTTP-з'єднань і, як наслідок, пришвидшення завантаження композитних цифрових ресурсів із внутрішніми залежностями.
5. Наведено програмне забезпечення для відтворення експериментів та можливість подальшого його розширення (підстратегії, пріоритизація, обмеження синхронних операцій за FPS, складніші залежності між ресурсами), яке уможливить підвищення адаптивності алгоритму до різних класів мультимедійних застосунків та подальше вдосконалення ефективності завантаження цифрових ресурсів.

References

1. Bang, J., Ha, R., & Cha, H. (2001). A web content scheduling for improved latency. *IEEE International Conference on Multimedia and Expo, 2001. ICME 2001*, 789–792. https://doi.org/10.1109/ICME.2001.1237840
2. Barford, P., & Crovella, M. (1999). A performance evaluation of hypertext transfer protocols. *SIGMETRICS 99: Proceedings of the 1999 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, pp. 188–197. https://doi.org/10.1145/301453.301560
3. Chang, T., Zhuang, Z., Velayutham, A., & Sivakumar, R. (2007). Client-side web acceleration for low-bandwidth hosts. *2007 Fourth International Conference on Broadband Communications, Networks and Systems* (*BROADNETS 07*), 932–941. https://doi.org/10.1109/BROADNETS.2007.4550537
4. Fenii, N. S., & Hrytsiuk, Y. I. (2020). Automation of the process of classification of text news from internet sites by neural network methods. *Scientific Bulletin of UNFU*, 30(4), 123–133. https://doi.org/10.36930/40300421
5. Fujinoki, H., Sanjay, M., & Shah, C. (2004). Web file transmission by object packaging – performance comparison with HTTP 1.0 and HTTP 1.1 persistent connection. *28th Annual IEEE International Conference on Local Computer Networks, 2003. LCN 03. Proceedings*. https://doi.org/10.1109/LCN.2003.1243114
6. Huang, J., Zhu, H., Liu, M., Zhang, T., & Wang, J. (2022). Achieving fast page load for websites across multiple domains. *Transactions on Emerging Telecommunications Technologies*, *33*(5). https://doi.org/10.1002/ett.4439
7. Kwok, Y., & Ahmad, I. (1999). Static scheduling algorithms for allocating directed task graphs to multiprocessors. *ACM Computing Surveys*, *31*(4), 406–471. https://doi.org/10.1145/344588.344618
8. Martin, R. C., & Martin, M. (2006). Agile principles, patterns, and practices in C#. *Pearson Education*. URL: https://www.google.com.ua/books/edition/Agile_Principles_Patterns_and_Practices/hckt7v6g09oC?gbpv=0
9. Marx, R., Quax, P., Faes, A., & Lamotte, W. (2017). Concatenation, Embedding and Sharding: Do HTTP/1 Performance Best Practices Make Sense in HTTP/2? *13th International Conference on Web Information Systems and Technologies*, pp. 160–173. https://doi.org/10.5220/0006364101600173
10. Marx, R., Wijnants, M., Quax, P., Faes, A., & Lamotte, W. (2018). Web Performance Characteristics of HTTP/2 vs HTTP/1.1. *Lecture notes in business information processing*, 87–114. https://doi.org/10.1007/978-3-319-93527-0_5
11. Nielsen, H. F., Gettys, Ja., Baird-Smith, A., Prudhommeaux, E., Lie, H. W., & Lilley, C. (1997). Network performance effects of HTTP/1.1, CSS1, and PNG. SIGCOMM 97: *Proceedings of the ACM SIGCOMM 97 conference on Applications, technologies, architectures, and protocols for computer communication*, pp. 155–166. https://doi.org/10.1145/263105.263157
12. Open source code of Phaser engine. URL: https://github.com/photonstorm/phaser
13. Open source code of PixiJS framework. URL: https://github.com/pixijs/pixijs
14. Sander, C., Blöcher, L., Wehrle, K., & Rüth, J. (2021). Sharding and HTTP/2 connection reuse revisited. *IMC 21: Proceedings of the 21st ACM Internet Measurement Conference*, pp. 292–301. https://doi.org/10.1145/3487552.3487832
15. Sander, C., Kunze, I., & Wehrle, K. (2022). Analyzing the influence of resource prioritization on HTTP/3 HOL blocking and performance. *COMSYS | RWTH Aachen University.* URL: https://www.comsys.rwth-aachen.de/publication/2022/2022_sander_analyzing-the-influence-of/
16. Song, M., Wang, N., & Haihong, E. (2018). Research on Optimization Algorithms for HTTP Maximum Concurrent Connection Restriction. *2018 International Conference on Information Systems and Computer Aided Education* (*ICISCAE*), pp. 493–498. https://doi.org/10.1109/ICISCAE.2018.8666833
17. Song, M., Wang, N., & Haihong, E. (2018). Research on Optimization Algorithms for HTTP Maximum Concurrent Connection Restriction. *2018 International Conference on Information Systems and Computer Aided Education* (*ICISCAE*), pp. 493–498. https://doi.org/10.1109/iciscae.2018.8666833
18. Sundaresan, S., Feamster, N., Teixeira, R., & Magharei, N. (2013). Community contribution award – Measuring and mitigating web performance bottlenecks in broadband access networks. *IMC 13: Proceedings of the 2013 conference on Internet measurement conference*, pp. 213–226. https://doi.org/10.1145/2504730.2504741
19. Torskyi, O. I., & Hrytsiuk, Y. I. (2025). Application of machine learning to enhance the efficiency of automated software testing. *Scientific Bulletin of UNFU*, 35(4), 142–149. https://doi.org/10.36930/40350416
20. Yu, J., Ding, Y., & Sato, H. (2025). DynTaskMAS: a dynamic task graph-driven framework for asynchronous and parallel LLM-based Multi-Agent systems. *Proceedings of the International Conference on Automated Planning and Scheduling*, *35*(1), pp. 288–296. https://doi.org/10.1609/icaps.v35i1.36130

Yu. Yu. Bilas, V. V. Riznyk
Lviv Polytechnic National University, Lviv, Ukraine

ALGORITHM FOR PARALLEL-SEQUENTIAL LOADING OF DIGITAL RESOURCES WITH CONTROL OF MAXIMUM SIMULTANEOUS LOADING OPERATIONS

A study was conducted on the features of organizing the process of loading digital resources in client web-oriented and hybrid multimedia applications under conditions of a limited number of simultaneous network connections. Existing approaches to loading digital resources were analyzed, in particular sequential, unlimited parallel loading (all at once), strategies with a fixed level of parallelism, as well as standard mechanisms for loading digital resources in popular interactive multimedia frameworks PixiJS and Phaser, and their limitations in the case of composite resources consisting of several interdependent files were identified. A model of the digital resource loading process is presented in the form of a dynamic task graph that includes synchronous and asynchronous operations and reflects the logical dependencies between sub-resources. An algorithm has been developed for parallel-sequential loading of digital resources with a controlled number of simultaneous HTTP connections, which ensures the combination of sequential initialization stages and parallel network operations without compromising data processing accuracy. A centralized parallelism level control mechanism has been implemented, allowing idle network connections to be avoided and at the same time not exceeding the set limit of simultaneous requests in the browser execution environment. An experimental comparison of the proposed approach with the above strategies was conducted on different types of multimedia data and execution platforms using multiple repeated measurements and statistical filtering of anomalous values. The experiments showed that the greatest improvements are achieved for datasets with composite resources, where the proposed approach reduces the loading time of digital resources by approximately 30 % compared to the standard PixiJS mechanism, by 9 % compared to the Phaser framework, and by 65 % compared to the sequential loading strategy. It has been found that the use of a dynamic task graph ensures more stable utilization of the available level of network parallelism and reduces the likelihood of HTTP connection downtime. The suitability of the proposed approach for use in client multimedia applications, where initialization time characteristics, predictability of the digital resource loading process, and efficient use of network resources are critical, has been demonstrated.

Keywords: media resources loading; multimedia applications; performance optimization; dynamic task graph; task parallelism.