DOI: 10.25140/2411-5363-2025-1(39)-161-169
УДК 004.415.53

Павло Володимирович Зінченко[1], Дмитро Борисович Мехед[2]

[1]аспірант кафедри інформаційних та комп'ютерних систем
Національний університет «Чернігівська політехніка» (Чернігів, Україна)
E-mail: zinchenkop@protonmail.com. **ORCID:** https://orcid.org/0009-0001-4573-6463. **ResearcherID:** MDS-7976-2025

[2]кандидат педагогічних наук, доцент кафедри кібербезпеки та математичного моделювання
Національний університет «Чернігівська політехніка» (Чернігів, Україна)
E-mail: d.mekhed@gmail.com. **ORCID:** https://orcid.org/0000-0003-3905-3620
ResearcherID: H-1751-2016. **SCOPUS Author ID**: 57193823626

ОПТИМІЗАЦІЯ ЗАПИТІВ GraphQL

Представлена у статті інформація має оглядовий характер. Сучасні виклики бізнесу потребують використання сучасних рішень для моніторингу та оптимізації програмних продуктів. Для ефективного моніторингу за допомогою GraphQL API необхідно проводити оптимізацію запитів. У статті виконано огляд деяких способів оптимізації запитів до GraphQL API: фільтрації та запиту декількох вузлів даних. На практичних прикладах використання GraphQL API популярного CDN постачальника Cloudflare було визначено, що фільтрація дозволяє зменшити обсяги отриманих даних, а запит декількох вузлів даних зменшує кількість запитів до API. Окремо підкреслюється важливість вивчення обмежень постачальників послуг та коректного формування фільтрів в запитах.

Ключові слова: *оптимізація; моніторинг; мова запитів; Cloudflare; API інтеграція.*
Табл.: 1. Бібл.: 8.

Актуальність теми дослідження. Представлена у статті інформація має оглядовий характер. Оптимізація та моніторинг сервісів, є важливими етапами життєвого циклу програмних продуктів [1]. Вони дозволяють забезпечувати стабільну роботу систем, своєчасно виявляти проблеми та усувати їх до того, як вони вплинуть на користувачів. Регулярний моніторинг допомагає отримати інформацію про продуктивність, доступність та ефективність сервісів, а оптимізація дозволяє підвищити їхню продуктивність, зменшити витрати на інфраструктуру та адаптувати системи до змін у навантаженні. У результаті ці процеси не лише підвищують якість продукту, але і сприяють економічній ефективності та довгостроковій стабільності бізнесу. Тому при розробці та підтримці програмних продуктів пошук та аналіз ефективних способів покращення моніторингу або оптимізації сервісів актуальний завжди.

Постановка проблеми. Продукти компаній можуть залежати від декількох зовнішніх постачальників послуг, наприклад CDN-постачальників, постачальників даних тощо. Зовнішні постачальники послуг, можуть надавати статистику використання їх сервісів і зазвичай роблять це через окремі вебсторінки. Цю статистику можна використовувати в цілях моніторингу сервісів компанії або для аналізу вузьких місць продуктів та подальшої оптимізації. Проте постає проблема — окремі вебсторінки, створюють незручності для компанії, особливо коли один продукт залежить від кількох таких постачальників. Перемикання між різними інтерфейсами та форматами даних ускладнює отримання цілісної картини й вимагає додаткового часу й зусиль. Це може призводити до втрати ефективності, помилок в аналізі та підвищення складності управління продуктом. Для вирішення цієї проблеми все частіше використовують централізовані рішення — комплексні моніторингові системи які об'єднують статистику з різних джерел у єдиний зручний інтерфейс за допомогою інтеграцій, спрощуючи процес аналізу. Однак навіть із такими системами постає інша проблема — такі інтеграції зазвичай є базовими, надаючи або багато зайвої інформації, або недостатньо інформації. До того ж ця інформація зазвичай не поєднана з бізнес-логікою продукту, що знижує їхню практичну цінність і потребує додаткового доопрацювання.

Компанії можуть розробляти власні інтеграції з використанням API, що дозволяє налаштувати моніторинг відповідно до специфіки продукту. Однак такий підхід має свої виклики. Здебільшого постачальники послуг надають REST API, який часто повертає

велику кількість непотрібної інформації, що ускладнює її обробку та потребує додаткових зусиль для фільтрації та структурування. Крім того, для отримання всіх необхідних даних доводиться робити багато запитів [2], що може суттєво навантажувати інтеграцію, особливо у випадках великого обсягу даних. Це також може призвести до додаткових витрат, пов'язаних з оплатою використання API, оскільки деякі постачальники стягують додаткову плату за понаднормову кількість запитів або високий обсяг переданих даних. Таким чином, власні інтеграції на основі REST API хоч і забезпечують більшу гнучкість поєднуючи статистику із бізнес-логікою продуктів компанії, але потребують ретельного планування, додаткових ресурсів, щоб уникнути зайвих витрат і знизити ризики перевантаження моніторинг системи. Часткове вирішення цих проблем — GraphQL API.

Аналіз останніх досліджень і публікацій. Одним із сучасних ефективних рішень для вирішення зазначених проблем є використання GraphQL API, якщо постачальник послуг його надає [3]. Як показало одне із досліджень «REST vs GraphQL: A Controlled Experiment» [2], ця технологія дозволяє ефективніше працювати із даними, зменшивши кількість запитів до API, а також вона простіша для розуміння в порівняні із REST API. Вибір цієї технології значно оптимізує роботу власних інтеграцій компанії та знижує навантаження на інструменти моніторингу та їх розробку, а також може допомогти уникнути зайвих витрат. В іншому дослідженні пропонують оптимальні методи розрахунку ціни виконання запитів до GraphQL API за допомогою ML (machine learning) як зі сторони сервера, так і зі сторони клієнта [4]. Водночас у науковому середовищі не знайшлось досліджень, статей, тез на тему оптимізації GraphQL запитів із погляду клієнта із практичними прикладами.

Тому **мета цієї статті** проаналізувати способи оптимізації GraphQL запитів з боку клієнта на основі практичних прикладів запитів до API постачальника CDN послуг Cloudflare [5].

Виділення недосліджених частин загальної проблеми. Аналіз наявних наукових робіт показав відсутність практичних прикладів способів оптимізації GraphQL запитів для складних, великих інформаційних систем постачальників послуг.

Виклад основного матеріалу. GraphQL — це мова запитів для API та середовище виконання, яке дозволяє клієнтам отримувати саме ті дані, які потрібні, без надлишкової інформації. Його основна перевага полягає в гнучкості, що дає змогу формувати запити під конкретні потреби, оптимізуючи роботу з даними [6].

Побудуємо простий graphql-запит для отримання базової статистики кількості запитів до доменної зони за допомогою curl:

```
{
 viewer {
  zones( filter: { zoneTag: "1234567890abcdef1234567890abcdef" } ) {
   httpRequestsAdaptiveGroups (
      filter: {
        datetime_geq: "2025-01-01T00:00:00Z"
        datetime_leq: "2025-01-02T00:00:00Z"
      }
   ) {
     count
     sum { edgeResponseBytes }
   }
  }
 }
}
```

У цьому запиті viewer і zones визначають область доступу, filter разом з zoneTag фільтрує дані для конкретної зони, httpRequestsAdaptiveGroups вузол повертає дані за останній день (за допомогою filter із визначенням початку та кінця проміжку часу для збору статистики) із

сумарною кількістю запитів (count) і переданих байтів (edgeResponseBytes). Побудований graphql-запит можна надіслати через команду curl, наприклад:

```
cat ./query.txt | tr -d '\n' | curl --silent \
https://api.cloudflare.com/client/v4/graphql \
--header "Authorization: Bearer <cloudflare_API_ключ>" \
--header "Content-Type: application/json" \
--data @-
```

де ./query.txt файл із підготовленим graphql-запитом та екранованими символами. У результаті такого API запиту отримаємо приблизно наступну відповідь:

```
{
 "data": {
  "viewer": {
   "zones": [ {
    "httpRequestsAdaptiveGroups": [ {
     "count": { "requests": 333222111 },
     "sum": { "edgeResponseBytes": 5444333222111 }
    } ]
   } ]
  }
 }
}
```

У результаті повертаються дані за період з 2025-01-01 до 2025-01-02 (1 день), де загальна кількість запитів до зони (requests) складає 333222111, а обсяг переданих даних (edgeResponseBytes) – 5444333222111 байтів. Ця інформація надає загальний огляд статистики зони, але не дозволяє детально аналізувати конкретні проблеми, пов'язані з роботою сервісів компанії, використанням їхніх ресурсів, географічними особливостями або налаштуванням кешування. Тому варто розглянути приклад graphql-запиту із додатковими параметрами, який нададасть глибше розуміння ситуації:

```
query { viewer {  zones( filter: { zoneTag: "1234567890abcdef1234567890abcdef" } ) {
 httpRequestsAdaptiveGroups (
    filter: {
     datetime_geq: "2025-01-01T00:00:00Z"
     datetime_leq: "2025-01-02T00:00:00Z"
    }
 ) {
  count
  sum { edgeResponseBytes }
  dimensions {
   clientRequestHTTPHost
   clientRequestPath
   clientRequestHTTPMethodName
   cacheStatus
   edgeResponseStatus
   userAgent
  }
 }
} } }
```

У цьому запиті додані параметри dimensions, які дозволяють згрупувати дані про кількість запитів та обсяг переданого трафіку за вказаними полями, а саме:

- clientRequestHTTPHost – FQDN (fully qualified domain name) до якого зверталися користувачі, наприклад www.company.com, app.company.com;
- clientRequestPath – шлях ресурсу до якого звертається клієнт, напр./app/auth;
- clientRequestHTTPMethodName – HTTP метод, який використовує користувач для запиту, наприклад GET, POST;
- cacheStatus – кеш статус, який повернув Cloudflare, напр. HIT, MISS, NONE та інші;
- edgeResponseStatus – HTTP код відповіді Cloudflare до користувача, напр. 200, 503;
- userAgent – рядок із характеристиками, за якими сервери та мережеві вузли можуть визначити тип застосунку, операційну систему, виробника та/або версію користувацького

агента, наприклад Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/51.0.2704.103 Safari/537.36.

У результаті отримаємо схожу відповідь:


```
{
 "data": {
  "viewer": {
   "zones": [ {
    "httpRequestsAdaptiveGroups": [ {
     "dimensions": {
      "clientRequestHTTPHost": "status.company.com"
      "clientRequestPath": "/"
      "clientRequestHTTPMethodName": "GET"
      "cacheStatus": "NONE"
      "edgeResponseStatus": "200"
      "userAgent": "Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/51.0.2704.103
Safari/537.36"
     }
     "count": { "requests": 233222111 },
     "sum": { "edgeResponseBytes": 3444333222111 }
    },
    {
     "dimensions": {
      "clientRequestHTTPHost": "app.company.com"
      "clientRequestPath": "/auth"
      "clientRequestHTTPMethodName": "POST"
      "cacheStatus": "NONE"
      "edgeResponseStatus": "200"
      "userAgent": "Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/51.0.2704.103
Safari/537.36"
     }
     "count": { "requests": 133222111 },
     "sum": { "edgeResponseBytes": 2444333222111 }
    },
    {
     "dimensions": {
      "clientRequestHTTPHost": "app.company.com"
      "clientRequestPath": "/auth"
      "clientRequestHTTPMethodName": "POST"
      "cacheStatus": "NONE"
      "edgeResponseStatus": "503"
      "userAgent": "Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/55.0.0000.00
Safari/537.36"
     }
     "count": { "requests": 22111 },
     "sum": { "edgeResponseBytes": 222111 }
    },
    <далі продовження списку>
    ]
   } ]
  }
 }
}
```


З використанням більш деталізованого запиту можна глибше проаналізувати дані та ідентифікувати потенційні проблеми в роботі сервісу, як у цьому прикладі, пов’язані з авторизацією. Вигадана проблема авторизації не є предметом аналізу в цій статті, але важливо зазначити, що висока деталізація даних, тобто велика кількість запитувальних полів, може призвести до іншої складності — великої кількості унікальних елементів у відповіді у вузлі httpRequestsAdaptiveGroups.

Велика кількість елементів у вузлі httpRequestsAdaptiveGroups може викликати проблеми. Одна із проблем виникає через обмеження Cloudflare GraphQL API, яке дозволяє повернути максимум 10000 елементів у межах одного вузла відповіді [7]. У разі перевищення цього ліміту частина даних не буде включена у відповідь, що може

спотворити результати аналізу. Основною причиною збільшення кількості елементів є велика кількість унікальних значень у деяких полях, зазначених у dimensions, таких як clientRequestHTTPHost, clientRequestPath і userAgent. Наприклад, якщо в запиті є 3 унікальних значення httpRequestsAdaptiveGroups, 5 різних шляхів у clientRequestPath і 100 унікальних значень userAgent (різні версії браузерів), то їх комбінація створює 3 × 5 × 100 = 1500 унікальних елементів. Ця кількість може легко зростати з додаванням нових полів або збільшенням кількості унікальних значень у кожному з них.

Якщо кількість елементів у відповіді перевищує встановлений ліміт у 10000, Cloudflare автоматично обрізає відповідь, залишаючи тільки дозволену кількість елементів. Це може призвести до втрати важливої інформації та ускладнити аналіз. Тому при розробці інтеграцій з API зовнішніх постачальників послуг необхідно враховувати ці обмеження.

Оптимізація. Залежно від специфіки поставленої задачі, яку має виконувати інструмент моніторингу, можливі різні варіанти використання. Кожен із цих варіантів визначає рівень деталізації та обсяг даних, необхідних для забезпечення ефективного контролю та аналізу. Основні випадки включають:

• Моніторинг окремого сервісу — цей підхід передбачає збір інформації про стан і продуктивність конкретного сервісу. Він актуальний у випадках, коли необхідно детально аналізувати роботу окремого компонента системи, наприклад, для виявлення проблем, аналізу продуктивності або оптимізації окремих процесів.

• Моніторинг набору сервісів — у цьому випадку інструмент моніторингу здійснює збір даних про кілька визначених сервісів, що входять до певної групи. Такий підхід зазвичай використовується для аналізу взаємодії між сервісами, відстеження загальної продуктивності окремих кластерів або компонентів, які мають спільну функціональність.

• Моніторинг усіх сервісів у межах зони — цей сценарій охоплює збір даних про всі сервіси, які функціонують у певній зоні або домені. Такий підхід дозволяє отримати повний огляд загального стану інфраструктури, виявити аномалії, розподілити ресурси або забезпечити безперервність роботи в масштабах всієї системи.

Кожен із цих варіантів вимагає чітко визначеного підходу до налаштування інтеграцій, відповідного рівня агрегації даних. Для вирішення зазначених завдань можна застосовувати фільтрацію та запит одразу декількох вузлів. Ці способи можна використовувати як окремо, так і в комбінації, залежно від потреб і специфіки моніторингових задач. Розглянемо кожен із цих способів.

Одним із найпростіших способів оптимізації GraphQL-запитів є **застосування фільтрації**. Замість того щоб отримувати дані про всі сервіси в зоні, можна сфокусуватися на конкретному об'єкті моніторингу, наприклад, окремому сервісі або відповідному FQDN (Fully Qualified Domain Name). Це значно знижує обсяг отримуваних даних, скорочує час обробки запитів і підвищує продуктивність інтеграції.

У наведених вище прикладах вже використовувалась фільтрація для визначення часового проміжку збору статистики за допомогою параметрів datetime_geq і datetime_leq. Подібним чином можливо фільтрувати дані за будь-яким доступним полем. Наприклад, якщо потрібно отримати інформацію про запити, що надійшли на певний домен, можна використовувати фільтр за полем clientRequestHTTPHost:

```
...
    httpRequestsAdaptiveGroups (
      filter: {
        datetime_geq: "2025-01-01T00:00:00Z"
        datetime_leq: "2025-01-02T00:00:00Z"
        clientRequestHTTPHost: "app.company.com"
      }
    )
...
```

У цьому прикладі фільтрація здійснюється за точним збігом значення поля fqdn із зазначеним у фільтрі. Однак можливості фільтрації не обмежуються лише точним збігом – для фільтрації можуть бути використані так звані оператори, що дозволяють гнучкіше налаштовувати запити. Поля для визначення часових проміжків вже демонструють використання операторів, де префікси _geq означає "greater or equal" (більше або дорівнює), а _leq – "lower or equal" (менше або дорівнює). Для текстових полів, таких як clientRequestHTTPHost, можна застосовувати оператори на кшталт _like, _in та інших операторів, залежно від функціонала постачальника послуг. Варто зазначити, що оператори, як і типи даних, є реалізацією постачальника послуг, а не частиною самого GraphQL [8]. Це означає, що набір доступних операторів і їх функціональність можуть відрізнятися залежно від обраного постачальника. Тому під час роботи з API важливо звертатися до документації, щоб коректно використовувати доступні інструменти фільтрації.

Іншим способом оптимізації, який дозволяє зменшити кількість GraphQL-запитів, є **запиту кількох вузлів одночасно**. Це дозволяє отримувати дані з різних частин API за один виклик або із тієї самої частини джерела для двох фільтрів одночасно за один виклик.

Наприклад, замість виконання кількох окремих запитів для збору даних для різних фільтрів, можна об’єднати ці запити в один. Цей підхід не лише оптимізує взаємодію із сервером, а й підвищує ефективність роботи клієнтської програми, зменшуючи затримки, пов’язані з обробкою кількох послідовних викликів.

```
query { viewer { zones( filter: { zoneTag: "1234567890abcdef1234567890abcdef" } ) {
   first_group: httpRequestsAdaptiveGroups (
     filter: {
       datetime_geq: "2025-01-01T00:00:00Z"
       datetime_leq: "2025-01-02T00:00:00Z"
       clientRequestHTTPHost: "app.company.com"
     }
   ) { count }
   another_group: httpRequestsAdaptiveGroups (
     filter: {
       datetime_geq: "2025-01-01T00:00:00Z"
       datetime_leq: "2025-01-02T00:00:00Z"
       clientRequestHTTPHost: "status.company.com"
     }
   ) { count }
} } }
```

До того ж цей спосіб оптимізації дозволяє отримувати більший обсяг даних за один GraphQL-запит, оскільки кожен окремий вузол повертає до 10000 елементів. Таким чином, із двох вузлів у межах одного запиту можна отримати до 20000 елементів одночасно.

Однак, при додаванні додаткових вузлів, якщо вони пов’язані із тією самою частиною API, важливо уважно налаштовувати фільтри, щоб дані, що відфільтровуються, не перекривалися. Нехтування цим правилом може призвести до дублювання даних, що ускладнює їх обробку та аналіз. Точне визначення фільтрів для кожного вузла є важливим аспектом для забезпечення коректності отриманих результатів.

Ознайомившись із двома способами оптимізації розглянемо результати оптимізації реальних GraphQL-запитів на реальному прикладі. Доменні імена сервісів змінені на видумані через NDA. Задача наступна: отримати кількість запитів до сервісу та кількість переданих байтів сервісом за кожен день, протягом останніх 30 днів. Не оптимізований GraphQL-запит виглядає наступним чином:

```
{ viewer { zones( filter: { zoneTag: "1234567890abcdef1234567890abcdef" } ) {
  group1: httpRequestsAdaptiveGroups (
    filter: {
      datetime_geq: "2025-01-01T00:00:00Z"
      datetime_leq: "2025-01-02T00:00:00Z"
    }
  ) { count sum { edgeResponseBytes } dimensions { clientRequestHTTPHost } }
} } }
```

Перша проблема полягає в тому, що цей запит потрібно буде виконати **30 разів**, (кожного разу міняючи дату на наступний день), через це швидкість отримання всіх даних в циклі сумарно дорівнює близько **10,3 секунди**. Друга проблема в тому, що кожна відповідь повертає інформацію про всі сервіси, тому кожного разу потрібно буде обробляти велику кількість інформації. Сумарно всі запити повернули **406 190 байтів** для обробки. Тепер застосуємо розглянуті способи оптимізації: фільтрацію та запит декількох вузлів:

```
{ viewer { zones( filter: { zoneTag: "1234567890abcdef1234567890abcdef" } ) {
 group1: httpRequestsAdaptiveGroups (
  filter: {
   datetime_geq: "2025-01-01T00:00:00Z"
   datetime_leq: "2025-01-02T00:00:00Z"
   clientRequestHTTPHost: "download.company.com"
  }
 ) { count sum { edgeResponseBytes } dimensions { clientRequestHTTPHost } }
 group2: httpRequestsAdaptiveGroups (
  filter: {
   datetime_geq: "2025-01-02T00:00:00Z"
   datetime_leq: "2025-01-03T00:00:00Z"
   clientRequestHTTPHost: "download.company.com"
  }
 ) { count sum { edgeResponseBytes } dimensions { clientRequestHTTPHost } }
...
 group15: httpRequestsAdaptiveGroups (
  filter: {
   datetime_geq: "2025-01-15T00:00:00Z"
   datetime_leq: "2025-01-16T00:00:00Z"
   clientRequestHTTPHost: "download.company.com"
  }
 ) { count sum { edgeResponseBytes } dimensions { clientRequestHTTPHost } }
} } }
```

У Cloudfalre існують обмеження на надмірне використання GraphQL API, тому оптимізований запит охоплює інформацію за 15 днів. Після оптимізації, для отримання такої самої кількості інформації потрібно буде виконати запит всього **2 рази** (по 15 днів у кожному). Загальний час виконання цих запитів займе **2,6 секунди** та необхідно буде обробити всього **4188 байтів** сумарно.

Як видно, якщо порівняти запити без оптимізації та запити із застосованими розглянутими в статті способами оптимізації – спостерігається загальний приріст у швидкості майже в **4 рази**, а кількість інформації для обробки зменшилась майже в **97 разів**. Проте варто розуміти, що для різних задач і даних – будуть зовсім різні результати. Для наочності додається порівняльна таблиця застосованих способів оптимізації.

Таблиця 1 – порівняльна таблиця способів оптимізації

Оптимізація запиту	Кількість запитів	Повернуто байт (сумарно для всіх запити), байт	Швидкість виконання (сумарно для всіх запитів), с
Без оптимізації	30	406190	~10,3
Тільки фільтрація	30	8850	~8,2
Тільки запит кількох вузлів	2	402822	~3,1
Фільтрація та запит кількох вузлів	2	4188	~2,6

Джерело: розроблено авторами.

Висновки. Оптимізація та моніторинг є важливими складовими життєвого циклу будь-якого продукту. Вони забезпечують стабільну роботу систем, допомагають швидко ідентифікувати проблеми та оперативно їх вирішувати, що підвищує якість послуг для кінцевих користувачів. У випадках, коли продукти використовують зовнішніх постачальників послуг, котрі надають статистику через GraphQL API, ефективна оптимізація запитів для отримання статистики використання цих постачальників послуг

стає важливою. Завдяки правильній побудові GraphQL-запитів можна зменшити обсяг отриманих даних, скоротити час виконання запитів і знизити витрати на використання API. Це дозволяє отримувати потрібну інформацію без перевантаження систем моніторингу зайвими даними.

Одним із найпростіших та водночас ефективних способів оптимізації GraphQL-запитів є використання фільтрації. За допомогою фільтрів можна чітко визначити, які саме дані потрібні, обмежуючи запит лише релевантною інформацією. Наприклад, фільтрація за часовими проміжками, доменами чи іншими параметрами дозволяє зосередитись на вузькому наборі даних, уникаючи отримання непотрібної інформації. Інший спосіб оптимізації — запит кількох вузлів одночасно. Це дозволяє поєднати дані з різних джерел або фільтрів у межах одного виклику, зменшуючи кількість запитів до API. Такий підхід особливо корисний, коли необхідно обробляти великі обсяги даних або працювати з кількома сегментами одночасно.

Водночас важливо враховувати технічні обмеження постачальників послуг, адже вони можуть впливати на кількість даних, які можна отримати за один запит. Наприклад, у Cloudflare GraphQL API існує ліміт на максимальну кількість елементів у межах одного вузла — до 10 000 і якщо кількість елементів у запиті перевищує цей ліміт, частина даних може буде обрізна, тобто відповідь буде неповна. Також під час планування інтеграцій із GraphQL API необхідно ретельно підбирати фільтри, уникаючи перетину даних, які створюють дублікацію елементів.

Розглянутий у статті реальний приклад оптимізації graphql-запитів підтвердив ефективність оптимізації. Результат оптимізації в прикладі полягає в пришвидшені отримання інформації майже в 4 рази, а кількість інформації для обробки зменшилась майже в 97 разів.

У підсумку, можна сказати, що оптимізація GraphQL-запитів не лише підвищує продуктивність моніторингових рішень, а й забезпечує точність і повноту зібраної інформації, що є дуже важливим для моніторингу продукту.

Список використаних джерел

1. Sai, K. (n.d.). DevOps Monitoring. Increase awareness during each stage of the delivery pipeline. *Atlassian*. https://www.atlassian.com/devops/devops-tools/devops-monitoring.

2. Brito, G., & Valente, M.T. (2020, March). REST vs GraphQL: A controlled experiment. In *2020 IEEE international conference on software architecture (ICSA)* (pp. 81-91). IEEE. https://doi.org/10.48550/arXiv.2003.04761.

3. Jones, D. (2023, 16 December). Emerging trends in graphql apis: the technology powering the future of data exchange. *Postman, Inc.* https://blog.postman.com/emerging-trends-graphql-apis-technology-future-of-data-exchange.

4. Mavroudeas, G., Baudart, G., Cha, A., Hirzel, M., Laredo, J. A., Magdon-Ismail, M., & Wittern, E. (2021). Learning graphql query costs (extended version). https://doi.org/10.48550/arXiv.2108.11139.

5. Nisenzoun, F. (2019, 12 hrudnia). Introducing the graphql analytics api: exactly the data you need, all in one place. *The Cloudflare Blog*. https://blog.cloudflare.com/introducing-the-graphql-analytics-api-exactly-the-data-you-need-all-in-one-place.

6. Byron, L. (2015, 14 veresnia). GraphQL: A data query language. *GraphQL*. https://graphql.org/blog/2015-09-14-graphql.

7. *GraphQL API - limits Â·cloudflare analytics docs. (n.d.). Cloudflare Docs*. https://developers.cloudflare.com/analytics/graphql-api/limits.

8. *Filtering Â. cloudflare analytics docs*. (n.d.). Cloudflare Docs. https://developers.cloudflare.com/analytics/graphql-api/features/filtering.

References

1. Sai, K. (n.d.). DevOps Monitoring. Increase awareness during each stage of the delivery pipeline. *Atlassian*. https://www.atlassian.com/devops/devops-tools/devops-monitoring.

2. Brito, G., & Valente, M.T. (2020, March). REST vs GraphQL: A controlled experiment. In *2020 IEEE international conference on software architecture (ICSA)* (pp. 81-91). IEEE. https://doi.org/10.48550/arXiv.2003.04761.

3. Jones, D. (2023, 16 travnia). Emerging trends in graphql apis: the technology powering the future of data exchange. *Postman, Inc.* https://blog.postman.com/emerging-trends-graphql-apis-technology-future-of-data-exchange.

4. Mavroudeas, G., Baudart, G., Cha, A., Hirzel, M., Laredo, J. A., Magdon-Ismail, M., & Wittern, E. (2021). Learning graphql query costs (extended version). https://doi.org/10.48550/arXiv.2108.11139.

5. Nisenzoun, F. (2019, 12 hrudnia). Introducing the graphql analytics api: exactly the data you need, all in one place. *The Cloudflare Blog*. https://blog.cloudflare.com/introducing-the-graphql-analytics-api-exactly-the-data-you-need-all-in-one-place.

6. Byron, L. (2015, 14 veresnia). GraphQL: A data query language. *GraphQL*. https://graphql.org/blog/2015-09-14-graphql.

7. *GraphQL API - limits Â·cloudflare analytics docs. (n.d.). Cloudflare Docs*. https://developers.cloudflare.com/analytics/graphql-api/limits.

8. *Filtering Â. cloudflare analytics docs*. (n.d.). Cloudflare Docs. https://developers.cloudflare.com/analytics/graphql-api/features/filtering.

Отримано 30.01.2025

UDC 004.415.53

Pavlo Zinchenko [1], Dmytro Mekhed [2]

[1]PhD Student at the Department of Information and Computer Systems
Chernihiv Polytechnic National University (Chernihiv, Ukraine)
E-mail: zinchenkop@protonmail.com. **ORCID:** https://orcid.org/0009-0001-4573-6463. **ResearcherID:** MDS-7976-2025

[2]PhD in Pedagogy, Associate Professor of the Department of cybersecurity and mathematical simulation
Chernihiv Polytechnic National University (Chernihiv, Ukraine)
E-mail: d.mekhed@gmail.com. **ORCID:** https://orcid.org/0000-0003-3905-3620
ResearcherID: H-1751-2016. **SCOPUS Author ID**: 57193823626

OPTIMIZATION OF GraphQL QUERIES

Presented in the article information is an overview. Many companies use several service providers to provide their services to customers. Sometimes it is not enough to have basic monitoring in centralized monitoring systems, so companies need to set up the custom API integration for monitoring. To collect REST API data is the main option, but usually leads to many requests and a huge amount of useless data. GraphQL API as modern replacement for REST API could solve these problems, but only with the correct optimization. For effective monitoring using the GraphQL API, it is necessary to optimize queries, because unoptimized queries could lead to monitoring tools overload and long timing of data processing. The purpose of the article is to review practical ways to optimize GraphQL API requests from the client part using the example of popular CDN provider Cloudflare. This article shows ways to optimize GraphQL API queries by "filtering" and "querying multiple data nodes". Based on examples of using the Cloudflare GraphQL API, these two optimization ways were manually analyzed. The "filtering" optimization allows reducing the amount of useless data received and avoiding data loss in some cases. The "querying multiple data nodes" optimization reduces the number of requests to the API and allows to get information from several parts of the API. Relevence of understanding limitations of service providers and correct formation of the GraphQL filters in queries are also analyzed within the article, because it could lead to the situation when data from the query will be duplicated. The article includes the actual example of optimizing GraphQL queries that confirms effective reviewed optimization methods — these methods allowed to speed up query execution and reduce the amount of information to be processed by several times. As a result, a correct GraphQL requests optimization increases effectiveness, accuracy and completeness of collected information.

Keywords: *optimization; monitoring; query; Cloudflare; API integration.*
Table: 1. References: 8.