JCSE Journal of Computing Science and Engineering

Regular Paper

Journal of Computing Science and Engineering,
Vol. 17, No. 4, December 2023, pp. 195-206

Performance Optimization of GraphQL API Through Advanced Object Deduplication Techniques: A Comprehensive Study

Budi Santosa*, Awang Hendrianto Pratomo, and Riski Midi Wardana
Department of Informatics, Universitas Pembangunan Nasional Veteran, Yogyakarta, Indonesia
dissan@upnyk.ac.id, awang@upnyk.ac.id, 123170035@student.upnyk.ac.id

Shoffan Saifullah*
Faculty of Computer Science, AGH University, Krakow, Poland;
Department of Informatics, Universitas Pembangunan Nasional Veteran, Yogyakarta, Indonesia
saifulla@agh.edu.pl, shoffans@upnyk.ac.id

Novrido Charibaldi
Department of Informatics, Universitas Pembangunan Nasional Veteran, Yogyakarta, Indonesia
novrido@upnyk.ac.id

Abstract

This research paper presents a comprehensive analysis of the performance enhancement achieved in GraphQL application programming interfaces (APIs) when using meticulous object deduplication implementation. By integrating advanced techniques into the GraphQL response mechanism, the data size exchanged between servers and clients can be significantly reduced. Rigorous testing against untreated and HTTP-compressed data validates the obtained results, highlighting the presence of substantial improvements across various performance metrics. The applied object deduplication method demonstrates gains in throughput, with a 0.33% increase observed in a 100-page test. Notably, response time analysis reveals enhancements of 11.04% (10 pages), 62.53% (20 pages), and an impressive 95.22% (100 pages). Meanwhile, parsing time evaluation showcases remarkable increases of 75.78% (10 pages), 276.38% (50 pages), and an even more exceptional 309.35% (100 pages). Comparative analysis against HTTP compression further validates the superiority of object deduplication in parsing time efficiency, demonstrating gains of 64.61% (10 pages), 193.76% (50 pages), and 218.07% (100 pages). While the throughput performance remains comparable, slight differences can be observed in response time, with a 0.66% increase (10 pages), a minor decrease of 0.12 (50 pages), and a modest decline of 1.45% (100 pages). This study fills in existing research gaps and provides empirical evidence of the benefits of object deduplication in enhancing GraphQL API performance, thus enabling the effective optimization of GraphQL APIs.

Category: Smart and Intelligent Computing

Keywords: GraphQL API; Performance enhancement; Object deduplication; Benchmarking; Data transmission

Open Access http://dx.doi.org/10.5626/JCSE.2023.17.4.195 http://jcse.kiise.org

This is an Open Access article distributed under the terms of the Creative Commons Attribution Non-Commercial License (http://creativecommons.org/licenses/by-nc/4.0/) which permits unrestricted non-commercial use, distribution, and reproduction in any medium, provided the original work is properly cited.

Received 14 October 2023; **Accepted** 17 December 2023

*Corresponding Author

Copyright © 2023. The Korean Institute of Information Scientists and Engineers

pISSN: 1976-4677 eISSN: 2093-8020

I. INTRODUCTION

GraphQL has rapidly emerged as a prominent query language and runtime for application programming interfaces (APIs) [1, 2], thus revolutionizing how data is requested and delivered in modern web and mobile applications [3]. Its flexible and intuitive nature allows clients to define precisely the data they require, making it an attractive choice for developers seeking efficient data retrieval [4]. However, as GraphQL APIs handle increasingly complex data and face more extensive client requests [5, 6], it has become paramount to ensure optimal performance.

Data duplication poses a common challenge when attempting to retrieve and filter data using specific categories from data sources, such as when searching and filtering products using specific store categories on an e-commerce website [7]. The presence of duplicate data increases the overall data size transmitted over the network, and the data transmission times that are consequently increased impact application performance [8]. Web APIs must exhibit good response times to ensure timely data presentation on the client side [9]. According to Everts [10], websites experiencing performance degradation (i.e., response times exceeding 4.4 seconds) incur an average hourly revenue loss of $4,100. Moreover, a case study on Akamai Company revealed that poorly performing websites experienced a permanent abandonment rate of 28% [10].

One of the advantages of GraphQL is its ability to define the requested data, thereby replacing multiple API calls in REST with a single GraphQL API call [11]. Due to the absence of limitations on requested data size by GraphQL clients, the flexibility that GraphQL offers to clients in defining requested data also opens up possibilities of data size inflation [12] and repetition of the same data [13], which lead to data redundancies.

In research by Sakamoto et al. [14], HTTP compression was shown to reduce network traffic size by more than 50%. HTTP compression effectively reduces the data size transmitted over the network, thereby expediting the transmission process. However, this method does not address the parsing of raw data into objects on the client side. According to a study by Li et al. [15], over 80% of the time required to query JSON data is dominated by the parsing process. The time-consuming nature of data parsing shows the need to develop efficient techniques to reduce parsing time.

According to Xia et al. [16], object deduplication offers advantages over traditional compression by eliminating redundant data at the chunk or file level, thus eliminating the need for byte-by-byte data comparison, and making the process more computationally efficient. Object deduplication can also be leveraged to reduce the data size to be parsed on the client side, as it manipulates data at the object level after parsing the raw data [17]. Therefore, the current research applies object deduplication to the GraphQL response to enhance the performance of the GraphQL API.

This study aims to comprehensively investigate the impact of advanced object deduplication techniques on the performance of GraphQL APIs. Object deduplication involves identifying and removing redundant data within the API response, resulting in a streamlined data transmission process between the server and client. Object deduplication optimizes both data transmission and parsing efficiency by minimizing data duplication, improving response times, and enhancing overall throughput.

To gauge the effectiveness of object deduplication, rigorous testing is conducted, and the results are compared against untreated data and data compressed using HTTP compression, which is a widely employed optimization method. The performance metrics analyzed are throughput, response time, and parsing time. This study aims to obtain invaluable insights into the benefits and implications of integrating object deduplication methods into GraphQL APIs through meticulous benchmarking and empirical analysis.

The findings of this research show that the implementation of object deduplication leads to noteworthy performance enhancements. The devised methodology demonstrates notable improvements in throughput, reduced response times, and enhanced parsing efficiency across various test scenarios. Further, a comparative analysis with HTTP compression underscores the advantages of object deduplication in optimizing parsing time, thus solidifying its standing as a promising performance optimization technique for GraphQL APIs.

This study significantly contributes to the burgeoning field of GraphQL API performance optimization by addressing prevalent research gaps and substantiating its conclusions with empirical evidence. The acquired insights are expected to serve as a guiding resource for developers and system architects, thus empowering them to make informed decisions regarding object deduplication techniques to augment the efficiency and responsiveness of their GraphQL APIs. Ultimately, the outcomes of this study hold immense potential to elevate the overall user experience and scalability of GraphQL-based applications, ultimately ensuring their continued prominence in the rapidly evolving landscape of web and mobile technologies.

II. METHODS

This section presents a comprehensive and systematic approach that can be used to assess the performance optimization of GraphQL APIs by implementing advanced object deduplication techniques. Object deduplication is a promising method for enhancing the efficiency and responsiveness of GraphQL APIs by eliminating redundant data within the API response, thus reducing the data size transmitted between the server and client [18]. The research architecture consists of two main components:

the GraphQL Client and the GraphQL Proxy. The GraphQL Client is responsible for conducting performance evaluations and measurements, while the GraphQL Proxy is the server application, handling client requests and implementing object deduplication and data compression as required. This architecture provides a robust framework that allows for the evaluation of the effectiveness of object deduplication in GraphQL APIs.

The current study is primarily focused on the implementation of object deduplication techniques within the GraphQL response. After retrieving the data from the source, the raw data is parsed into objects, and deduplication is applied to remove duplicate objects before the response is transmitted to the client. By manipulating the data at the object level, this research aims to efficiently eliminate duplicate data, thus reducing response size and improving overall API performance.

To comprehensively evaluate the impact of object deduplication, this study compares three distinct architectural models: Architecture 1 (no treatment), Architecture 2 (HTTP compression), and Architecture 3 (object deduplication). Key performance metrics [19], such as throughput, response time, and parsing time, are analyzed through rigorous testing. This comparison between these different architectural models provides valuable insights into the benefits and implications of implementing object deduplication techniques in GraphQL APIs.

In the subsequent sections, this study presents detailed performance evaluations and statistical analysis to substantiate the effectiveness of object deduplication. Through robust statistical methods and careful interpretation of the results, this Methods section contributes to the validity and reliability of the research findings, with the goal of facilitating informed decision-making among developers and system architects in terms of the optimization of GraphQL API performance. This Methods section establishes a solid foundation for assessing the impact of object deduplication on GraphQL APIs, thus leading to valuable conclusions about and potential directions for advancement in GraphQL API optimization.

A. Data Collection

To ensure the validity and comprehensiveness of this study, a systematic and meticulous data collection process is used to gather relevant and diverse GraphQL API responses. The data collection methodology is designed to encompass various aspects of data retrieval and scenarios that are commonly encountered in real-world GraphQL API implementations [20]. The following steps outline the data collection process [21]:

1) Identification of relevant API endpoints: The first step involves carefully identifying and selecting API endpoints that represent typical data retrieval operations performed by GraphQL APIs. The endpoints are chosen to cover various data types, queries, and use cases, and they ultimately provide a representative sample of the API's functionality and performance characteristics.
2) API request and response capture: Using appropriate tools and techniques, API requests are sent to the identified endpoints, and the corresponding responses are captured. The data collection system records the raw HTTP responses, including headers and payloads, to ensure the existence of a sufficiently comprehensive dataset for analysis.
3) Data diversity and sample size considerations: To ensure data diversity, the data collection process considers different GraphQL queries, including queries with various complexities, nested fields, and other data sizes. This approach is intended to include responses that span a broad spectrum of complexities and user scenarios, thus allowing for a thorough evaluation of the object deduplication methods' performance. The sample size is also carefully determined to provide adequate statistical significance and minimize potential bias in the analysis. A sufficiently large dataset is collected to derive meaningful insights and robust conclusions from the evaluation.
4) Data preprocessing and cleaning: The collected data undergoes preprocessing and cleaning to ensure data consistency and eliminate noise or irrelevant information. Data preprocessing includes removing duplicate entries, handling missing values, and transforming data into a standardized format. This step is a crucial aspect in preparing the data for further analysis and facilitating accurate performance evaluation.
5) Data validation and quality assurance: To maintain the integrity and quality of the dataset, a comprehensive validation process is conducted. In particular, the data is cross-validated against GraphQL API responses to ensure accuracy and consistency. Any inconsistencies or discrepancies are carefully addressed and rectified to ensure the reliability of the data for subsequent analysis.
6) Ethical Considerations: During the data collection process, ethical considerations and data privacy protocols are strictly adhered to. In this process, our research team has ensured that user privacy is respected and that all collected data is anonymized to protect user identities and sensitive information. The data is used solely for research purposes and handled in compliance with applicable data protection regulations.

By employing a rigorous and systematic data collection methodology, this research aims to build a high-quality dataset that reflects real-world scenarios and accurately represents the performance characteristics of GraphQL

APIs. The resulting dataset provides a solid foundation for evaluating the effectiveness of object deduplication methods in optimizing GraphQL API performance, and it is expected to help obtain valuable insights into enhancing the efficiency and responsiveness of GraphQL-based applications.

B. Architecture Design

The design of the architecture plays a crucial role in the implementation of the object deduplication techniques and the optimization of the performance of GraphQL APIs.

This process involves carefully arranging components and integrating object deduplication methods into the existing GraphQL infrastructure. The architecture design for this study (Fig. 1) comprises two main details: the GraphQL Client and the GraphQL Proxy.

GraphQL Client: The GraphQL Client serves as the application that is responsible for conducting tests and measurements on the variables being examined [22]. It is designed to send GraphQL queries and requests to the GraphQL Proxy, thus simulating real-world scenarios and interactions between clients and the GraphQL API [23]. The GraphQL Client can take various forms, including mobile applications, web applications, or desktop applications, thus offering versatility in performance evaluations across different platforms.

GraphQL Proxy: The GraphQL Proxy is the server application, and it mediates between the GraphQL Client and the actual Github GraphQL API [24]. It acts as an intermediary that receives requests from the client, processes them, and forwards them to the GitHub GraphQL API. The primary objectives of the GraphQL Proxy are to implement object deduplication techniques and data compression before transmitting the data to the client.

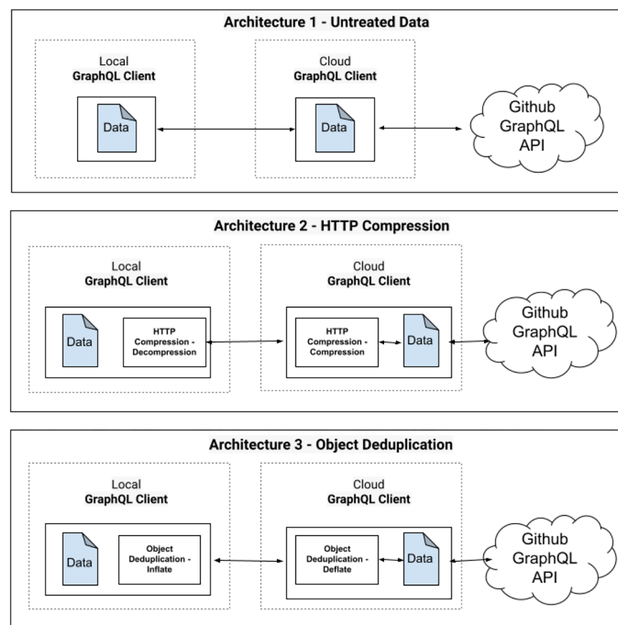

Fig. 1. Proposed system architecture and comparison among no-treatment, HTTP compression, and object deduplication.

Object deduplication implementation: The implementation of object deduplication occurs within the GraphQL Proxy. Before the API response is sent to the client, the GraphQL Proxy systematically identifies and eliminates redundant data objects [25]. Object deduplication significantly reduces the data size transmitted over the network, thus improving data transmission efficiency and response times.

Data deduplication: Data deduplication is a crucial technique in modern data management systems that aim to improve storage efficiency and optimize data transmission over networks. It involves identifying and eliminating duplicate data, reducing redundancy, and optimizing data storage capacity [26]. Through the removal of duplicate data, data deduplication saves storage space and minimizes data transfer times, thereby improving performance and reducing network bandwidth consumption. The present research explicitly applies data deduplication to the GraphQL API response with a focus on optimizing the data transmitted from the server to the client. The deduplication process is carried out at the object level after parsing the raw data, which allows for efficient manipulation of data objects and minimizes the data size processed on the client side. The data deduplication process [27] can be broadly outlined as follows:

- Chunking: The data is divided into smaller chunks or segments. Each chunk typically contains a portion of the overall data.
- Fingerprinting: Each chunk is transformed into a unique fingerprint or hash value based on its content. This fingerprinting process helps efficiently identify chunks that are identical or similar to each other.
- Indexing: The generated fingerprints are used to create an index or lookup table that facilitates the identification and grouping of duplicate chunks. This index allows for efficient detection of duplicate data across the entire dataset.
- Compression (optional): While compression is not an inherent part of the data deduplication process, it is sometimes used to further reduce the data size. Compression algorithms can eliminate additional redundancies and compact the data, thereby leading to even more efficient data transmission.

The effectiveness of data deduplication in reducing data size depends on the level of data duplication present in the dataset. As shown in the research findings obtained by Xia et al. [16], data duplication levels vary across

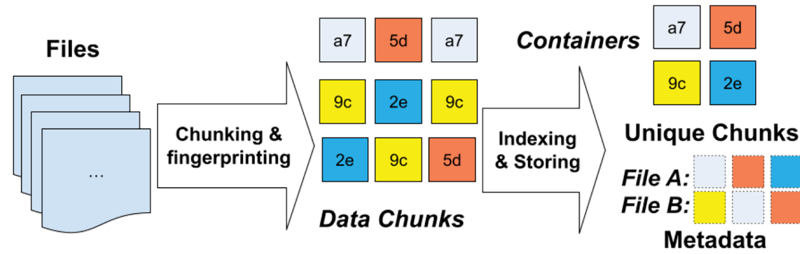

Fig. 2. Data deduplication process.

different workloads and environments (Fig. 2). For instance, file-level deduplication on Microsoft's internal users was shown to achieve a reduction ratio of about 21%, meaning that duplicate data accounted for only 21% of the original data size. By contrast, chunk-level deduplication across users at an 8-kB chunk size achieved a significantly higher reduction ratio of about 68%.

The benefits of data deduplication extend beyond storage optimization. By reducing the amount of data transmitted over the network, data deduplication can lead to faster retrieval and improved response times for GraphQL API requests. Furthermore, data deduplication has a crucial impact on parsing time, as it directly influences the time required to process data on the client side. GraphQL APIs can enhance performance by implementing data deduplication techniques and providing a more responsive and efficient user experience. The present research will comprehensively evaluate the effectiveness of data deduplication in enhancing the performance of GraphQL APIs through a series of experiments and analyses. The findings of this evaluation are expected to provide valuable insights into the benefits of applying object deduplication techniques to GraphQL API responses and their implications for overall performance optimization.

Data compression: The GraphQL Proxy is also equipped with data compression capabilities. Once the data is successfully retrieved from the GitHub GraphQL API, it is compressed using HTTP compression techniques. Data compression further optimizes data transmission, thus reducing network traffic and expediting data delivery to the client [28].

Architecture models: This research considers three distinct architectural models (Fig. 1) for evaluation:

- Architecture 1 (no treatment): This baseline architecture represents the control scenario where the data undergoes no treatment, i.e., neither object deduplication nor HTTP compression. It serves as the reference point for comparison against the other architectures while providing insights into the raw GraphQL API performance without the use of any optimization techniques.
- Architecture 2 (HTTP compression): In this model, HTTP compression is applied to the data. After successfully fetching data from the GitHub GraphQL API, the GraphQL Proxy compresses the data before sending it to the client over the network. This architecture serves as a benchmark for comparing the performance of object deduplication against a widely used compression technique.
- Architecture 3 (object deduplication): In this architecture, object deduplication is implemented on the data to be studied. The GraphQL Proxy identifies and eliminates redundant data objects before transmitting the data to the client over the network. This architecture focuses on assessing the performance improvements achieved by the object deduplication techniques. The proposed architecture design aims to comprehensively evaluate object deduplication techniques and their impacts on GraphQL API performance. By implementing object deduplication within the GraphQL Proxy, this study aims to optimize data transmission, reduce response times, and enhance the overall efficiency of GraphQL-based applications.

C. Performance Analysis

In this section, we present a detailed analysis of the performance metrics used to evaluate the impact of object deduplication on GraphQL API performance. The performance analysis involves measuring key metrics [29, 30] related to throughput, response time, and parsing time under various test scenarios.

1) Throughput measurement: Throughput is a critical performance metric that assesses the number of API requests served per unit of time. It provides insights into the system's capability to handle concurrent requests and its overall processing efficiency. To measure throughput, load tests are conducted in which the number of concurrent API requests is gradually increased while monitoring the response times and success rates. The throughput results are recorded and compared across different scenarios, including GraphQL APIs with and without object deduplication.
2) Response time evaluation: Response time represents the time taken by the GraphQL API to respond to a client request, including the time spent on processing

the request and generating the response. Performance testing involves executing predefined queries with varying complexities and sizes. The response times are measured for each query type and analyzed under different workloads and data loads. The response time data is statistically analyzed to identify the performance improvements achieved through object deduplication.

3) Parsing time assessment: Parsing time refers to the duration taken by the client-side to parse the received GraphQL response and convert it into usable objects. Parsing performance is evaluated by executing queries that return a substantial amount of data and measuring the time taken to process and parse the response. The obtained parsing times are compared between GraphQL APIs with and without object deduplication to gauge the impact that the deduplication technique has on parsing efficiency.

D. Performance Evaluation

The performance evaluation aims to assess the impact of the object deduplication method on the performance of the GraphQL API. To provide a comprehensive analysis, the following performance metrics are meticulously measured for each test scenario:

1) Throughput: Throughput is a fundamental performance metric that quantifies the number of successful requests completed by the GraphQL API within a specified time frame. It reflects the system's processing capacity and its ability to handle a high volume of incoming requests. Throughput is measured in requests per unit of time and typically represented in the form of requests per second (RPS). For each architecture model (untreated data, HTTP compression, and object deduplication), the throughput is calculated by tracking the number of successfully processed API requests over a fixed time duration. A higher throughput indicates that the GraphQL API has an enhanced capability to efficiently handle a greater number of client requests.
2) Response time: Response time measures the duration taken by the GraphQL API to process a client request and deliver the corresponding response back to the client. It is a critical metric that reflects the system's responsiveness and directly impacts user experience. Response time is typically expressed in milliseconds (ms). To evaluate the response time, the GraphQL Client sends multiple requests representing different test scenarios to the GraphQL API. The response time is recorded as the time taken from when each request is sent until the complete response is received. By analyzing the response times for each test scenario and architecture model, we can gauge the impact of object deduplication on improving response time.
3) Parsing time: Parsing time refers to the time required to parse the raw API response data into structured objects on the client-side. As GraphQL API responses are often in JSON format, parsing data can be a time-consuming process, particularly for large and complex responses. To measure parsing time, the client-side application records the time taken to convert the received JSON data into objects, thus preparing it for further processing and rendering. By comparing the parsing times across different architecture models, including Untreated Data, HTTP Compression, and Object Deduplication, the influence that object deduplication has on parsing efficiency can be determined.

III. RESULTS AND DISCUSSION

The performance evaluation aimed to compare the effectiveness of three different architectures: Architecture 1 (no treatment), Architecture 2 (HTTP compression), and Architecture 3 (object deduplication), in terms of their abilities to improve GraphQL API performance. The evaluation encompassed throughput, response size, response time, and parsing time comparisons. Testing was conducted while varying the page sizes among 10, 50, and 100 pages and using a GraphQL client to send multiple requests to the GraphQL server during a 5-minute testing period. The performance evaluation results are presented in Table 1, and the comparison of the response data from the three architectures is illustrated in Fig. 3.

The response data reflects the impact of three distinct architectural approaches—object deduplication, HTTP compression, and no treatment—on response sizes across varying page sizes. As can be seen in Fig. 3, object deduplication leads to a notable reduction in response sizes compared to no treatment, which is particularly evident with smaller page sizes. For instance, at a page size of 10, object deduplication results in a response size of 1796 bytes whereas no treatment yields a considerably larger response size of 8,978. This trend persists as the page size increases, with object deduplication consistently outperforming no treatment. Interestingly, HTTP compression also significantly reduces response sizes, albeit not quite to the extent of the reductions achieved by object deduplication. For example, at a page size of 10, HTTP compression results in a response size of 952 bytes. The effectiveness of both object deduplication and HTTP compression is particularly pronounced at smaller page sizes, thus suggesting their potential utility in optimizing data transfer efficiency. The results underscore the efficiency gains achieved by employing data optimization techniques. Object deduplication and HTTP compression promise to reduce response sizes and enhance overall system performance.

Table 1. Comparative analysis of performance testing: throughput, response size, response time, and parsing time

Model	Page size	Average			
		Response size (byte)	Response time (ms)	Parsing time (ms)	Throughput (req/s)
Architecture 1 (no treatment)	10	8,978	450.39	0.17	1
	50	43,335	720.88	0.71	1
	100	64,566	866.70	1.04	0.9967
Architecture 2 (HTTP compression)	10	952	408.29	0.16	1
	50	1,232	442.98	0.55	1
	100	1,400	437.53	0.81	1
Architecture 3 (object deduplication)	10	1,796	405.63	0.09	1
	50	5,031	443.53	0.18	1
	100	7,110	443.96	0.25	1

Architecture	Page Size	Response Size (byte)	ResponseData
Object Deduplication	10	1796	{“data”:{“repository”{“__typename”:”Repository”,”id”:”MDEwOIJk {“__typename”:”User”:”id”:”MDQ6VXNIcjM2zE20DIx”}}}},{“node
	50	5031	{“data”:{“repository”{“__typename”:”Repository”,”id”:”MDEwOIJk {“__typename”:”User”:”id”:”MDQ6VXNIcjM2zE20DIx”}}}},{“node {“__typename”:”User”:”id”:”MDQ6VXNIcjM2zE20DIx”}}}},{“node {“__typename”:”User”:”id”:”MDQ6VXNIcjM2zE20DIx”}}}},{“node
	100	7110	{“data”:{“repository”{“__typename”:”Repository”,”id”:”MDEwOIJk {“__typename”:”User”:”id”:”MDQ6VXNIcjM2zE20DIx”}}}},{“node {“__typename”:”User”:”id”:”MDQ6VXNIcjM2zE20DIx”}}}},{“node {“committer”:{“user”:{}__typename”:”User”,”id”:”MDQ6VXNIcjM2
HTTP Compression	10	952	H4sIAAAAAAAAA+2Tb2/aMBDGv0qU11BCmCoWqdraQf9pWddCC
	50	1232	H4sIAAAAAAAAA+3T7U7bSBSA4VuJ/JuQxKwQVRt6ZIC1WZpCV
	100	1400	H4sIAAAAAAAAA+3bUW/bNhDA8a9i6DmObWUIUgHFli5ukmJe1jj
No Treatment	10	8978	{“data”:{“repository”{“__typename”:”Repository”,”id”:”MDEwOIJk {“id”:”MDEwOIVzZXJTdGFOdXMyMzAzMjEZ”,”emoji”:”house:”, betadev”,”octicon”:”organization”}]},”isBountyHunter”:false,”isCa [{“message”:”Member of @permanent betadev”,”octicon”:”organ Enginer\r\n”,”company”:null,”email”:””,”hasSponsorsListing”:false v=4”,”login”:”riskimidiw”,”bio”:”Student and Software Engineer\r\n’

Fig. 3. Comprehensive comparison of response data among three architectures: Architecture 1 (no treatment), Architecture 2 (HTTP compression), and Architecture 3 (object deduplication).

A. Throughput Testing Results

Throughput testing measures the number of successful requests the server completes per unit of time. A higher throughput indicates better performance, as it reflects that the server successfully processed more client requests. Based on the research results presented in Table 1, the performance improvement or decline of Architecture 3 (object deduplication) was calculated and compared to those of Architecture 1 (no treatment) and Architecture 2 (HTTP compression), with the results presented in Table 2. The throughput testing results are visualized in diagram form in Fig. 4, which displays a comparative view of throughput performance among all architectures.

Throughput testing measured the number of successful requests the server completed per unit of time, thus reflecting the system's processing capacity. The results showed that all three architectures demonstrated similar throughput performance during testing with 10 and 50 pages, achieving a throughput of 1 request per second for each scenario. However, during testing with 100 pages, Architecture 1 (no treatment) showed a slight decline in throughput compared to Architecture 3 (object deduplication), resulting in a throughput of approximately 0.997 RPS.

The slight decline in throughput for Architecture 1 (no treatment) during testing with 100 pages can be attributed

Table 2. Throughput performance comparison

Model	Comparison architecture model	Page size	Throughput (%)	
			Increase	Decrease
Architecture 3 (object deduplication)	Architecture 1 (no treatment)	10	0	0
		50	0	0
		100	0.33	0
Architecture 3 (object deduplication)	Architecture 2 (HTTP compression)	10	0	0
		50	0	0
		100	0	0

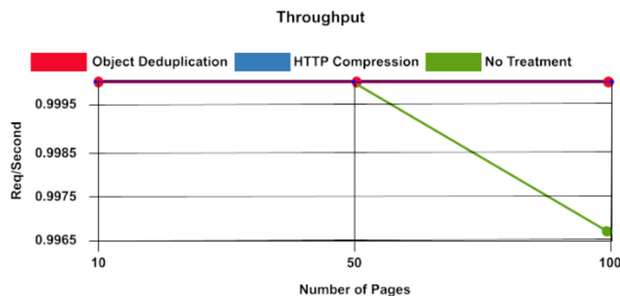

Fig. 4. Throughput testing results.

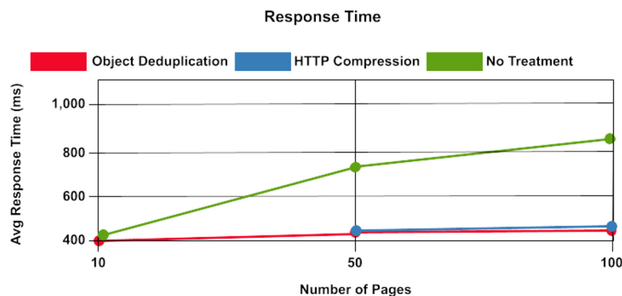

Fig. 5. Response time testing results.

to the increase in the size of the data that needed to be sent by the server for 100 pages. This resulted in longer processing times and reduced the number of successful requests, ultimately affecting the throughput for Architecture 1. On the other hand, Architecture 3 (object deduplication) showed no significant difference in throughput compared to Architecture 2 (HTTP compression) in all three testing scenarios. This similarity is attributed to the fact that the data compression in Architecture 2 helped maintain a similar data size to that of Architecture 3, thus ensuring comparable performance. Based on the findings from Fig. 4, the experimental results indicate that object deduplication and HTTP compression exhibit similar outcomes. In the graph displaying testing results, the data points for object deduplication and HTTP compression, which are respectively represented by red and blue lines, can be seen to overlap.

B. Response Time Testing Results

Response time testing measured the time taken for a client request to be processed and the response to be received from the server. A lower response time indicates better performance, as it means that the client receives the response faster. The results in Table 3 were used to calculate the percentage increase or decrease in the response time of Architecture 3 (object deduplication) compared to those in Architecture 1 (no treatment) and Architecture 2 (HTTP compression). The comparison results are presented in Table 3, and a visual representation is shown in Fig. 5.

The response time testing revealed that Architecture 1 (no treatment) exhibited the worst performance among all three architectures for all three testing scenarios. This can be attributed to the larger data size sent by the server to the client in this architecture, thus resulting in a longer time for the client to receive the response. On the other hand, Architecture 2 (HTTP compression) showed a comparable response time to Architecture 3 (object deduplication) for testing with 10 and 50 pages, which was again attributed to data compression reducing the data size transmitted over the network in Architecture 2. However, during testing with 100 HTTP compression showed a slight increase in response time compared to Architecture 3 (object deduplication).

C. Parsing Time Testing Results

Parsing time testing measured the time required to parse the API response data into objects on the client side. A lower parsing time indicates better performance, as it means that the client can process the data more efficiently. The results from Table 1 were used to calculate the percentage increase or decrease in parsing time of Architecture 3 (object deduplication), which were then compared to those of Architecture 1 (no treatment) and Architecture 2 (HTTP compression). The comparison

Table 3. Response time performance comparison

Model	Comparison architecture model	Page size	Response time (%)	
			Increase	Decrease
Architecture 3 (object deduplication)	Architecture 1 (no treatment)	10	11.04	0
		50	62.53	0
		100	95.22	0
Architecture 3 (object deduplication)	Architecture 2 (HTTP compression)	10	0.66	0
		50	0	0.12
		100	0	1.45

Table 4. Parsing time performance comparison

Model	Comparison architecture model	Page size	Parsing time (%)	
			Increase	Decrease
Architecture 3 (object deduplication)	Architecture 1 (no treatment)	10	75.78	0
		50	276.38	0
		100	309.35	0
Architecture 3 (object deduplication)	Architecture 2 (HTTP compression)	10	64.61	0
		50	193.76	0
		100	218.07	0

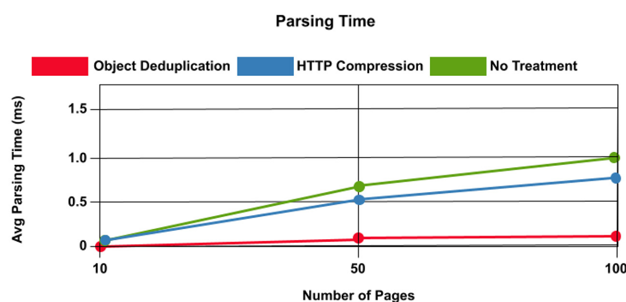

Fig. 6. Parsing time testing results.

results are presented in Table 4 and visualized in Fig. 6.

D. Discussion of Findings

The research outlined in this paper has involved conducting a meticulous analysis of the performance enhancements in GraphQL APIs achieved through the strategic implementation of object deduplication. Object deduplication, which is an essential technique integrated into the GraphQL response mechanism, significantly reduces the exchanged data size between servers and clients. Rigorous testing against untreated and HTTP-compressed data validates the efficacy of object deduplication, as it is shown to achieve marked improvements across multiple performance metrics. The application of the object deduplication method leads to noteworthy gains in throughput, with a 0.33% increase observed in a 100-page test, thus highlighting its efficiency in processing client requests and enhancing overall system throughput.

Further, the analysis of response time reveals substantial enhancements of 11.04% (10 pages), 62.53% (50 pages), and an impressive 95.22% (100 pages), thus signifying the effectiveness of object deduplication in expediting the processing of client requests and improving API responsiveness. The results of the evaluation of parsing time efficiency further underscore the benefits of object deduplication, as they demonstrate significant increases of 75.78% (10 pages), 276.38% (50 pages), and an even more exceptional 309.35% (100 pages). Comparative analysis against HTTP compression validates the superiority of object deduplication in parsing time efficiency, as it consistently outperforms HTTP compression by 64.61% (10 pages), 193.76% (50 pages), and 218.07% (100 pages). While the throughput performance remains comparable, minor differences in response time indicate nuanced considerations between object deduplication and HTTP compression based on specific performance requirements. Overall, this research contributes empirical evidence of the pivotal role played by object deduplication in optimizing GraphQL APIs, thus substantiating its practical significance and filling in critical research gaps.

The results of parsing time testing demonstrated that Architecture 3 (object deduplication) outperformed the other architectures in all three testing scenarios. This can be attributed to its deduplication technique, which resulted in a smaller data size for parsing. By contrast, Architecture 1 (no treatment) required the client to parse the entire response data, leading to a considerably longer parsing time. While Architecture 2 (HTTP compression) compressed the data before sending it to the client, the parsing time in that architecture was not significantly different from that of Architecture 3 (object deduplication), indicating that the need to decompress the data before parsing did not have a major effect in Architecture 2. The results of the performance evaluation indicated that Architecture 3 (object deduplication) offers significant benefits in each of response time and parsing time compared to both Architecture 1 (no treatment) and Architecture 2 (HTTP compression). By deduplicating data on the server side, Architecture 3 reduces the response size, which consequently reduces the time required for processing and parsing. As a result, it outperforms the other architectures, particularly in scenarios where data deduplication can be applied effectively.

IV. CONCLUSION

This research aimed to explore avenues for enhancing GraphQL API performance by investigating three distinct

architectural approaches: Architecture 1 (no treatment), Architecture 2 (HTTP compression), and Architecture 3 (object deduplication). By evaluating key performance metrics—including throughput, response size, response time, and parsing time—across varying page sizes (10, 50, and 100 pages), this study provides valuable insights. The findings underscore the significance of architectural choices in influencing GraphQL API performance. Among the architectures studied, Architecture 3 (object deduplication) emerged as a standout performer, particularly excelling in response time and parsing time. Leveraging the benefits of object deduplication, it effectively mitigated data redundancy, ultimately resulting in streamlined parsing processes and quicker response times. Architecture 2 (HTTP compression) demonstrated competitive throughput and response time metrics, although it incurred slightly extended parsing times due to the need for data decompression pre-parsing. Conversely, Architecture 1 (no treatment) yielded the poorest results, particularly in terms of response time and parsing time, which were attributed to its untreated nature, leading to inflated data sizes and culminating in slower responses as well as extended parsing times.

The study's implications extend to future research possibilities, including studies exploring combined architectural techniques, analyses of diverse data structures and request patterns, real-world scenario testing, and scalability assessment. From a practical perspective, developers and system architects can leverage these conclusions to make informed decisions in attempting to optimize GraphQL API performance. Through the judicious use of techniques like object deduplication and HTTP compression, tailored architectural choices, and comprehensive testing, GraphQL APIs can be designed to deliver efficient responsiveness and an enhanced user experience.

ACKNOWLEDGMENTS

The authors express their gratitude to UPN "Veteran" Yogyakarta, especially the Department of Information Engineering and LPPM, who assisted in preparing this article for publication. We also thank the Faculty of Computer Science, AGH University of Krakow, who supported this publication.

Conflict of Interest(COI)

The authors have declared that no competing interests exist.

REFERENCES

1. E. Wittern, A. Cha, J. C. Davis, G. Baudart, and L. Mandel, "An empirical study of GraphQL schemas," in *Service-Oriented Computing*. Cham, Switzerland: Springer, 2019, pp. 3-19. https://doi.org/10.1007/978-3-030-33702-5_1
2. A. Lawi, B. L. Panggabean, and T. Yoshida, "Evaluating GraphQL and REST API services performance in a massive and intensive accessible information system," *Computers*, vol. 10, no. 11, article no. 138, 2021. https://doi.org/10.3390/computers10110138
3. R. Khan and A. Noor Mian, "Sustainable IoT sensing applications development through GraphQL-based abstraction layer," *Electronics*, vol. 9, no. 4, article no. 564, 2020. https://doi.org/10.3390/electronics9040564
4. T. Diaz, F. Olmedo, and E. Tanter, "A mechanized formalization of GraphQL," in *Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs*, New Orleans, LA, USA, 2020, pp. 201-214. https://doi.org/10.1145/3372885.3373822
5. M. Obert and V. Buzek, "Industry trends impacting SAP UI technologies," in *SAP UI Frameworks for Enterprise Developers*. Berkeley, CA: Apress, 2023, pp. 237-268. https://doi.org/10.1007/978-1-4842-9535-9_6
6. G. Sagar and V. Syrovatskyi, "System design: architecting robust, scalable, and modular applications," in *Technical Building Blocks*. Berkeley, CA: Apress, 2022, pp. 105-168. https://doi.org/10.1007/978-1-4842-8658-6_3
7. W. Shin, J. Park, T. Woo, Y. Cho, K. Oh, and H. Song, "e-clip: large-scale vision-language representation learning in e-commerce," in *Proceedings of the 31st ACM International Conference on Information & Knowledge Management*, Atlanta, GA, USA, 2022, pp. 3484-3494. https://doi.org/10.1145/3511808.3557067
8. J. Ni, X. Lin, and X. S. Shen, "Toward edge-assisted Internet of Things: from security and efficiency perspectives," *IEEE Network*, vol. 33, no. 2, pp. 50-57, 2019. https://doi.org/10.1109/MNET.2019.1800229
9. A. R. Breje, R. Gyorodi, C. Gyorodi, D. Zmaranda, and G. Pecherle, "Comparative study of data sending methods for XML and JSON models," *International Journal of Advanced Computer Science and Applications*, vol. 9, no. 12, pp. 198-204, 2018. https://doi.org/10.14569/ijacsa.2018.091229
10. T. Everts, *Time Is Money: The Business Value of Web Performance*. Sebastopol, CA: O'Reilly Media Inc., 2016.
11. O. Hartig and J. Perez, "Semantics and complexity of GraphQL," in *Proceedings of the 2018 World Wide Web Conference*, Lyon, France, 2018, pp. 1155-1164. https://doi.org/10.1145/3178876.3186014
12. S. Cheng and O. Hartig, "LinGBM: a performance benchmark for approaches to build GraphQL servers," in *Web Information Systems Engineering – WISE 2022*. Cham, Switzerland: Springer, 2022, pp. 209-224. https://doi.org/10.1007/978-3-031-20891-1_16
13. J. G. Ogboada, V. I. E. Anireh, and D. Matthias, "A model for optimizing the runtime of GraphQL queries," *International Journal of Innovative Information Systems & Technology Research*, vol. 9, no. 3, pp. 11-39, 2021.
14. Y. Sakamoto, S. Matsumoto, S. Tokunaga, S. Saiki, and M. Nakamura, "Empirical study on effects of script minification and HTTP compression for traffic reduction," in *Proceedings of 2015 3rd International Conference on Digital Information, Networking, and Wireless Communications (DINWC)*, Moscow, Russia, 2015, pp. 127-132. https://doi.org/10.1109/DINWC.2015.7054230
15. Y. Li, N. R. Katsipoulakis, B. Chandramouli, J. Goldstein, and D. Kossmann, "Mison: a fast JSON parser for data analytics,"

Proceedings of the VLDB Endowment, vol. 10, no. 10, pp. 1118-1129, 2017. https://doi.org/10.14778/3115404.3115416
16. W. Xia, H. Jiang, D. Feng, F. Douglis, P. Shilane, Y. Hua, M. Fu, Y. Zhang, and Y. Zhou, “A comprehensive study of the past, present, and future of data deduplication,” *Proceedings of the IEEE*, vol. 104, no. 9, pp. 1681-1710, 2016. https://doi.org/10.1109/JPROC.2016.2571298
17. P. A. D. S. N. Wijesekara and S. Gunawardena, “A comprehensive survey on knowledge-defined networking,” *Telecom*, vol. 4, no. 3, pp. 477-596, 2023. https://doi.org/10.3390/telecom4030025
18. Z. Zhao, “Build a live news application with Next.js 13,” M.S. thesis, Metropolia University of Applied Sciences, Helsinki, Finland, 2023. https://www.theseus.fi/handle/10024/793229
19. E. Zeydan and J. Mangues-Bafalluy, “Recent advances in data engineering for networking,” *IEEE Access*, vol. 10, pp. 34449-34496, 2022. https://doi.org/10.1109/ACCESS.2022.3162863
20. C. Wang, L. Marini, C. L. Chin, N. Vance, C. Donelson, P. Meunier, and J. T. Yun, “Social media intelligence and learning environment: an open source framework for social media data collection, analysis and curation,” in *Proceedings of 2019 15th International Conference on eScience (eScience)*, San Diego, CA, USA, 2019, pp. 252-261. https://doi.org/10.1109/eScience.2019.00035
21. J. Ohme, T. Araujo, L. Boeschoten, D. Freelon, N. Ram, B. B. Reeves, and T. N. Robinson, “Digital trace data collection for social media effects research: APIs, data donation, and (screen) tracking,” *Communication Methods and Measures*, 2023. https://doi.org/10.1080/19312458.2023.2181319
22. G. Brito and M. T. Valente, “REST vs GraphQL: a controlled experiment,” in *Proceedings of 2020 IEEE International Conference on Software Architecture (ICSA)*, Salvador, Brazil, 2020, pp. 81-91. https://doi.org/10.1109/ICSA47634.2020.00016
23. E. Frigard, “GraphQL vs. REST: a comparison of runtime performance,” M.S. thesis, Linnaeus University, Vaxjo, Sweden, 2022.
24. M. V. De F. Borges, L. S. Rocha, and P. H. M. Maia, “MicroGraphQL: a unified communication approach for systems of systems using microservices and GraphQL,” in *Proceedings of the 10th IEEE/ACM International Workshop on Software Engineering for Systems-of-Systems and Software Ecosystems*, Pittsburgh, PA, USA, 2022, pp. 33-40. https://doi.org/10.1145/3528229.3529381
25. O. Debauche, S. Mahmoudi, P. Manneback, and F. Lebeau, “Cloud and distributed architectures for data management in Agriculture 4.0: review and future trends,” *Journal of King Saud University-Computer and Information Sciences*, vol. 34, no. 9, pp. 7494-7514, 2022. https://doi.org/10.1016/j.jksuci.2021.09.015
26. Q. He, Z. Li, and X. Zhang, “Data deduplication techniques,” in *Proceedings of 2010 International Conference on Future Information Technology and Management Engineering*, Changzhou, China, 2010, pp. 430-433. https://doi.org/10.1109/FITME.2010.5656539
27. S. Bansal and P. C. Sharma, “Classification criteria for data deduplication methods,” in *Data Deduplication Approaches*. London, UK: Academic Press, 2021, pp. 69-96. https://doi.org/10.1016/B978-0-12-823395-5.00011-2
28. H. F. Nielsen, J. Gettys, A. Baird-Smith, E. Prud'hommeaux, H. W. Lie, and C. Lilley, “Network performance effects of HTTP/1.1, CSS1, and PNG,” in *Proceedings of the ACM SIGCOMM'97 conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, Cannes, France, 1997, pp. 155-166. https://doi.org/10.1145/263105.263157
29. D. Bermbach and E. Wittern, “Benchmarking web API quality-revisited,” *Journal of Web Engineering*, vol. 19, no. 5-6, pp. 603-646, 2020. https://doi.org/10.13052/jwe1540-9589.19563
30. M. I. Ladan, “Web services metrics: a survey and a classification,” *Journal of Communication and Computer*, vol. 9, no. 7, pp. 824-829, 2012.

Budi Santosa

Budi Santosa received master's degree in software engineering from Informatics Department, Institute of Technology Bandung, Indonesia in 2001. He joined Department of Informatics, Faculty of Industrial Engineering, Universitas Pembangunan Nasional "Veteran" Yogyakarta, Indonesia in May 2003. His research interests are in software engineering, geographic information system, and geo-informatics.

Awang Hendrianto Pratomo http://orcid.org/0000-0002-9760-0285

Awang Hendrianto Pratomo received bachelor program at Department Informatics Engineering (S.T), Idonesian Islamic University, and master program at Information Technology (M.T), Department of Electrical Engineering, Gadjah Mada University. He received Ph.D. in System Science and Management, Faculty of Information Science and Technology, from Universiti Kebangsaan Malaysia. His research interests include robotic, artificial intelligent systems, and software engineering. A lot of papers, articles, and tutorials have been published in many conference proceedings, scientific journal, magazine, and newspaper, in national and international level. Current research on development autonomous mobile robot specially on robotics soccer, vision system in robotics soccer, firefighting robots, and industrial robotics systems.

Riski Midi Wardana

Riski Midi Wardana received a bachelor's degree in Department of Informatics from Universitas Pembangunan Nasional "Veteran" Yogyakarta, Indonesia, in 2023. He is currently working as a senior software engineer at Grab. His research interests include algorithms, software engineering, and cloud computing.

Shoffan Saifullah https://orcid.org/0000-0001-6799-3834

Shoffan Saifullah received a bachelor's degree in informatics engineering from Universitas Teknologi Yogyakarta, Indonesia, in 2015 and a master's degree in computer science from Universitas Ahmad Dahlan, Yogyakarta, Indonesia, in 2018. He is a lecturer at Universitas Pembangunan Nasional "Veteran" Yogyakarta, Indonesia. His research interests include image processing, computer vision, and artificial intelligence. He is currently a Ph.D. student at AGH University of Krakow, Poland, with a concentration in the field of artificial intelligence (bio-inspired algorithms), image processing, and medical image analysis.

Novrido Charibaldi https://orchid.org/0000-0002-7709-6181

Novrido Charibaldi was born in Palembang, Indonesia. He received his S.Kom. (Bachelor of Informatics Engineering) from Sekolah Tinggi Sains dan Teknologi Indonesia (ST.INTEN), Bandung, Indonesia, in 1994, and his M.Kom. (Master of Computer Science) from Universitas Gadjah Mada, Yogyakarta, Indonesia, in 2001. He received doctoral degree in Computer Science from Universitas Gadjah Mada, Yogyakarta, Indonesia, in 2019. His main research interests are in feature extraction, machine learning, and pattern recognition for odor detection.