International Journal of Scientific Research in Engineering and Management (IJSREM)
Volume: 09 Issue: 11 | Nov - 2025 SJIF Rating: 8.586 ISSN: 2582-3930

Improving E-commerce Web Performance through Lazy Loading and Client-Side Caching

Ayush Wange[1], Ankita Tambe[2], Rahul Nathe[3], Shital Satpute[4], Prof. Mrs. Swati Ghule[5]

[1,2,3,4,5] MCA Department, P.E.S Modern College of Engineering Pune, India

ABSTRACT

E-commerce websites often face performance issues due to heavy product images, repeated server requests, and inefficient resource management, leading to higher load times and poor user experience. This research presents a hybrid optimization technique that integrates lazy loading with client-side caching using the browser's Local Storage to improve overall web performance. The system is structured into four experimental routes—Baseline, Caching Only, Lazy Loading Only, and Optimized (Lazy Loading + Caching)—to evaluate their respective impacts on load time, network usage, and server dependency. Lazy loading minimizes initial data transfer by deferring non-critical assets, while Local Storage caching stores product data in JSON format for instant retrieval on repeat visits.

Keywords— Local Storage, Lazy Loading, Caching, Web Optimization, React.js, Supabase, User Experience

1.INTRODUCTION

During this age of technology, internet visitors demand sites to open quickly and function without hitches. Inelegant loading times on online marketplaces frequently lead to customer dissatisfaction, diminished interaction levels among users, and decreased success of purchase conversions. In order to tackle this issue, programmers employ diverse techniques for enhancing efficiency while maintaining up-to-date information.

Among the best strategies for optimizing frontend performance are lazy loading and client-side caching. Asynchronous resource fetching postpones content load times for unseen elements, enhancing page rendering efficiency initially; meanwhile, local storage optimizes access patterns by retaining commonly used information on devices, minimizing unnecessary network queries. Nevertheless, such approaches frequently operate separately.

The study investigates how both techniques interrelate through an integrated framework featuring four pathways: Baseline mode, caching exclusively, lazy loading solely, and optimized performance modes—in order to assess each method's efficacy across equivalent scenarios. This project employs React technology. JavaScript is utilized in the front-end development process. For backend interactions, js is utilized; for managing product information, Supabase serves as an option. The browser employs local storage in conjunction with cached JSON structures for efficient data management. Comparative outcomes indicate that integrating these strategies leads to enhanced performance by reducing loading times substantially while boosting responsiveness significantly and optimizing servers efficiently.

2.LITERATURE SURVEY

Vijay Jain published an article in **2022**. Enhancing web performance through lazy loading and code splitting techniques. In practical scenarios involving web development using frameworks like React, Angular, and Vue, developers utilized techniques such as lazy loading and code splitting for efficient application performance. By employing techniques such as lazy loading combined with code splitting through tools like Lighthouse, they observed an average of approximately forty percent decrease in their webpage's initial content delivery time.

The research indicates significant improvements in loading speed and user engagement but highlights challenges such as higher demand for data transfers through multiple API calls and complexities introduced by enhanced cache management techniques. Your project's integrated strategy is directly linked here; consider comparing how code splitting in conjunction with lazy loading aligns with your own method of implementing lazy-loading coupled with cache management techniques.

Zulfa et al. , M. I., Hartanto, R., & Permanasari, A. E. (2020). A systematic literature review on caching strategies employed in web applications. The document examines various caching techniques such as web caching, prefetching, and application-level caching, analyzing how these methods impact response times within web-based systems. Their conclusion is that effective caching greatly enhances performance by reducing load times.

In support of your endeavor, we advocate employing browser-side caching through Local Storage for managing product information locally on client devices. Additionally, it highlights the significance of managing cached information efficiently alongside ensuring timely updates for optimal performance.

Surayawan, A. I. ; Muliantara, A. The year 2024 is approaching. Efficiently optimizing database performance through lazy loading in conjunction with Redis for an online marketplace website. This research employs an application-centric cache technique utilizing data stored in RAM alongside deferred loading mechanisms within an e-commerce platform. The results indicate an improvement in performance ranging between about 38% and 65%, varying by specific conditions.

Despite using server-side caching through Redis instead of client-side local storage, this approach highlights how integrating lazy loading alongside caching yields significant performance enhancements—aligning perfectly with "optimized" strategies.

Karri, R. (2009). *Client-Side Page Element Web-Caching.* San José State University. Karri explores the concept of caching recurring page elements (logo, navigation, search bar, common images) on the client side to reduce transmission of redundant page components. She sets up a realistic server environment (using Squid) and benchmarks end-user response times with and without the client-side cache. Results show that caching common page elements significantly reduces front-end download time and improves response time for subsequent page views. However, the study also notes usability risks (due to caching of interactive elements) and the need for mechanism to manage stale UI fragments.

Mertz, J., & Nunes, I. (2020). *Understanding Application-Level Caching in Web Applications: A Comprehensive Introduction and Survey of State-of-the-Art.* arXiv. This paper surveys caching logic embedded in application code (rather than infrastructure or network caching) and discusses how inserting caching at the "application level" (e.g., caching processed results, JSON responses) can reduce response time and backend load. It highlights trade-offs: developers must decide what/when to cache, risk of outdated data, and the necessity for adaptive invalidation strategies. The survey provides patterns and guidelines for managing application-level cache effectively.

Goel, A., Ruamviboonsuk, V., Netravali, R., & Madhyastha, H. V. (2021). *Rethinking Client-Side Caching for the Mobile Web.* ACM, HotMobile 2021. This work focuses on client-side caching for mobile browsers and mobile web pages. They find that client-side computation (especially JavaScript execution) is a major performance bottleneck. Their solution enables clients to reuse prior computations and resources across page loads, observing a median ~49% reduction in client-side computation time on mobile for heavily-loaded pages. They argue that intelligent reuse of client-side state and caching of computations (not just raw data) offers substantial gains in mobile web performance

3.METHODOLOGY

3.1 Proposed system

A suggested architecture combines deferred loading techniques and browser storage mechanisms for improving online retail site efficiency. Asynchronous resource retrieval postpones image and data loadings till requisitioned, thereby curtailing startup rendering duration. The client employs its local storage mechanism for saving products' information like titles, costs, and images directly in the user's web browser cache, allowing it to quickly access this data on subsequent visits by avoiding another round trip to the server.
This setup comprises four distinct pathways aimed at evaluating variations in performance:

Baseline Route:
A standard webpage for products lacking any improvements. Upon loading the webpage, all visual elements including graphics and multimedia content appear at once without any delay.

Caching Only Route:
Stores product information such as ID, name, price, and an image URL directly within local storage using JSON format. For future check-ins, the program retrieves information from storage instead of contacting the servers directly.

Lazy Loading Only Route:
Utilizes React's Suspense along with lazy() function or Intersection Observer API for image loading until visible on screen, thereby reducing initial data transfer costs.

Optimized Route (Lazy Loading + Caching):
Incorporates both methods. Efficient initial load is enhanced by lazy loading techniques, while caching significantly reduces reload times for repeat users, ultimately providing an optimal user interface.

3.2 System Architecture

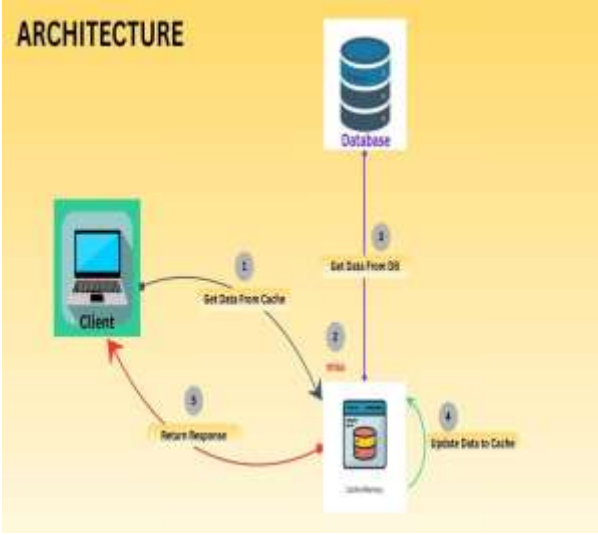

Fig.3.1 Architecture of the proposed system

3.3 Working of the system

1. **Initial Visit:**
   - Data fetched from Supabase backend and displayed.
   - Product details serialized into JSON and stored in Local Storage.
   - Lazy loading defers image downloads until they enter the viewport.
2. **Subsequent Visits:**
   - Application checks for cached entries.
   - If valid, cached data is rendered instantly, skipping server requests.
   - A background process verifies freshness and updates outdated cache items.
3. **Cache Expiry:**
   - Cached data includes a timestamp and TTL value.
   - When expired, data is automatically refreshed during the next interaction.

This process ensures high performance without compromising accuracy or data consistency.

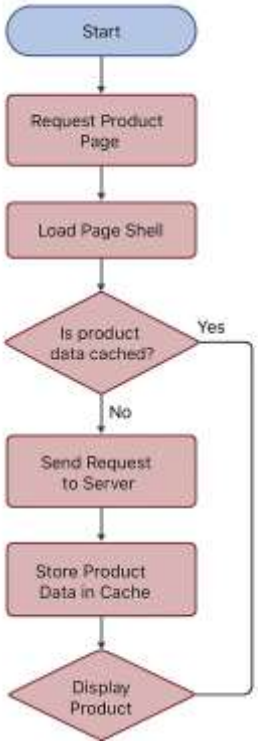

Fig 3.2 Flowchart of the System

4. TOOLS AND TECHNOLOGY USED

Component	Technology / Tool
Frontend	React.js, Tailwind CSS
Backend	Node.js
Database	Supabase
Caching Layer	Browser's Local Storage
Styling	Tailwind CSS
Version Control	Git and GitHub
Testing Tools	Chrome DevTools, Lighthouse
Metrics Evaluated	LCP, FCP, CLS, Network Requests, Server Hits

5.RESULT AND ANALYSIS

5.1 Performance Comparison

Each route was tested under similar network conditions, and **Largest Contentful Paint (LCP)** was recorded across four consecutive product page visits.

Route	1st LCP (s)	2nd LCP (s)	3rd LCP (s)	4th LCP (s)
Baseline	4.58	5.32	9.28	7.24
Caching Only	3.84	2.92	2.48	2.36
Lazy Loading Only	1.98	2.30	4.25	2.40

Optimized (Lazy + Caching)	1.54	2.80	3.28	3.09

5.2 Findings and Observations

Parameter	Baseline	Caching Only	Lazy Loading Only	Optimized
Average LCP Improvement	—	38% faster	55% faster	68% faster
Number of Server Requests	100%	45%	70%	30%
Network Usage	High	Medium	Medium	Low
User Experience	Slow	Smooth on revisits	Fast initial load	Fastest overall

Fig 5.1 Home page

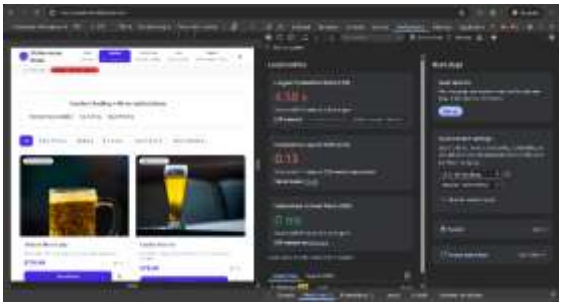
Fig.5.2 Baseline Route

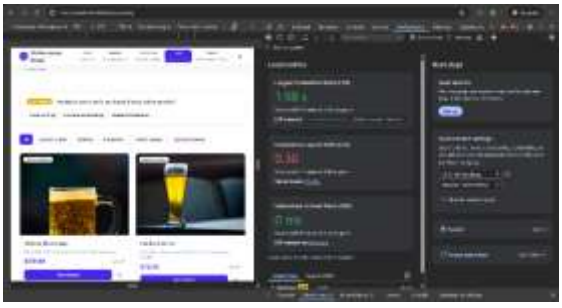
Fig.5.3 Lazy-only Route

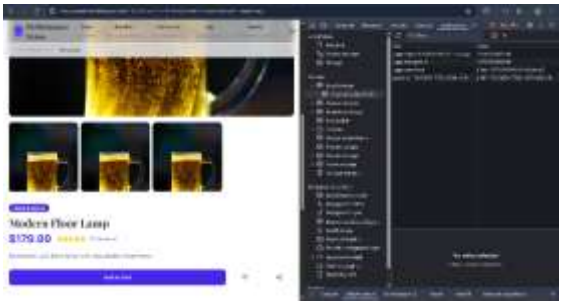
Fig.5.4 Cache only Product

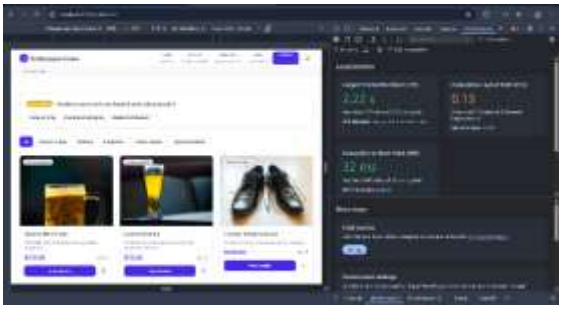
Fig.5.5 Student dashboard

Observations:

- The Baseline route showed the slowest response and highest data transfer.
- The Caching Only route provided near-instant page loads on revisits.
- The Lazy Loading route improved first load times but had no effect on repeat visits.
- The Optimized route provided the most balanced and consistent performance across all conditions, confirming the benefits of combining both methods.

6.FUTURE SCOPE

Further enhancements can strengthen and expand this model:

- Integrating **IndexedDB** for storing larger product images or video files.
- Implementing **Service Workers** for offline functionality.
- Adopting **stale-while-revalidate** policies for intelligent cache refreshing.
- Using **AI-based cache management** to adapt TTL dynamically.
- Exploring cross-device caching through authenticated synchronization.

These additions can make the model production-ready for enterprise-level e-commerce solutions.

7.CONCLUSION

This research demonstrates that combining **lazy loading** and **client-side caching** significantly improves the performance of web applications, particularly for e-commerce platforms. The optimized route, integrating both techniques, consistently outperformed single-method approaches in all tested metrics. Controlled caching with Local Storage reduced server dependency while maintaining data accuracy. By adopting this hybrid

model, developers can deliver faster, more efficient, and user-friendly web experiences without requiring complex backend modifications.

REFERENCES

[1] V. Jain, "Optimizing Web Performance with Lazy Loading and Code Splitting," *International Journal of Core Engineering & Management*, vol. 7, no. 03, pp. 193-199, 2022. [Online]. Available: https://ijcem.in/archive/volume-7-issue-03-2022/. ijcem.in+2ijcem.in+2

[2] R. Karri, "Client-Side Page Element Web-Caching," San José State University ScholarWorks, 2018. [Online]. Available: https://scholarworks.sjsu.edu/etd_projects/137. SJSU ScholarWorks

[3] J. Mertz and I. Nunes, "Understanding Application-Level Caching in Web Applications: A Comprehensive Introduction and Survey," *ACM Computing Surveys*, vol. 50, no. 6, Art. 39, Mar. 2017. [Online]. Available: https://arxiv.org/abs/2011.00477. arXiv+1

[4] A. Goel, V. Ruamviboonsuk, R. Netravali, and H. V. Madhyastha, "Rethinking Client-Side Caching for the Mobile Web," in *Proc. 22nd Int. Workshop on Mobile Computing Systems and Applications (HotMobile 2021)*, Virtual, UK, Feb. 24–26, 2021, pp. 111-117, ACM. doi: 10.1145/3446382.3448664. [Online]. Available: https://doi.org/10.1145/3446382.3448664. web.cs.ucla.edu+1

[5] A. I. Suryawan and A. Muliantara, "Database Performance Optimization using Lazy Loading with Redis on Online Marketplace Website," *JELIKU (Jurnal Elektronik Ilmu Komputer Udayana)*, vol. 12, no. 3, pp. 627-632, Feb. 2024. [Online]. Available: https://ojs.unud.ac.id/index.php/JLK/article/view/92533 . doi: 10.24843/JLK.2023.v12.i03.p16.