Scientific publishing house

Professional Bulletin

Information Technology and Security

Issue №3/2025

A scientific journal for the best specialists in the industry. Inside: original works on AI, data analysis, cloud technologies and IT innovations.

INDEXATIONS

Google Scholar · International Scientific Indexing · Academic Resource Index ResearchBib · Научная электронная библиотека eLIBRARY.RU · CyberLeninka · CiteFactor Academic Scientific Journals

ISSN 3100-444X

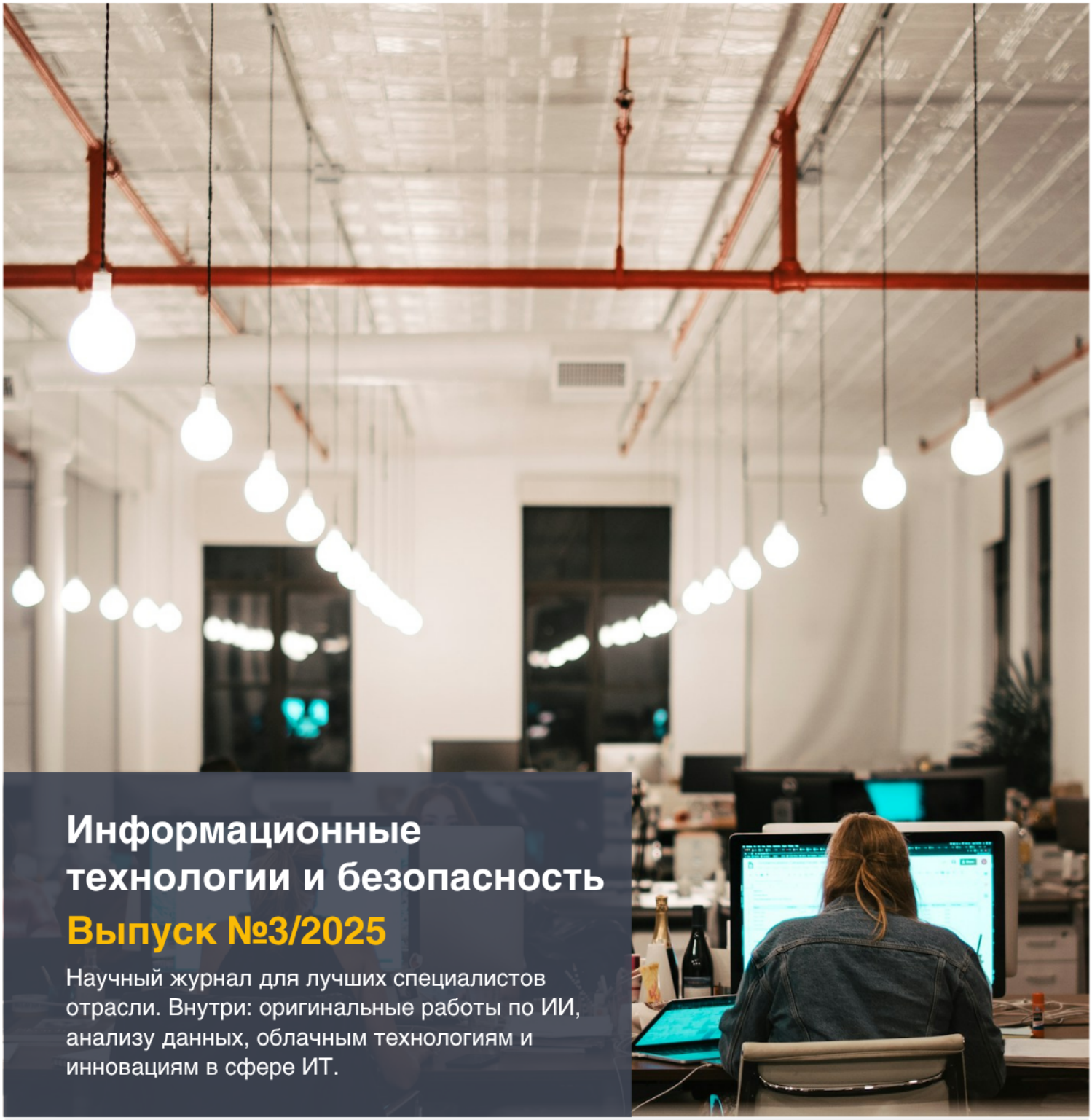
support@professionalbulletinpublisher.com

professionalbulletinpublisher.com/

Brasov, Sat Sanpetru, Comuna Sanpetru, Str. Sfintii Constantin si Elena, nr. 6

Научное издательство

Профессиональный Вестник



Информационные технологии и безопасность

Выпуск №3/2025

Научный журнал для лучших специалистов отрасли. Внутри: оригинальные работы по ИИ, анализу данных, облачным технологиям и инновациям в сфере ИТ.

ИНДЕКСАЦИИ ЖУРНАЛА

Google Scholar · International Scientific Indexing · Academic Resource Index ResearchBib · Научная электронная библиотека eLIBRARY.RU · CyberLeninka · CiteFactor Academic Scientific Journals

ISSN 3100-444X

support@professionalbulletinpublisher.com

professionalbulletinpublisher.com/

Brasov, Sat Sanpetru, Comuna Sanpetru, Str. Sfintii Constantin si Elena, nr. 6

The scientific publishing house «Professional Bulletin»
Journal «Professional Bulletin. Information Technology and Security»

Professional Bulletin. Information Technology and Security is a professional scientific journal. The publication in it is recommended to practitioners and researchers who seek to find solutions to real-world problems and share their experiences with the professional community. The publication in journal is suitable for those specialists who work and actively develop advanced IT solutions, such as AI, blockchain, big data technologies and others.

The journal reviews all incoming materials. The review is double-blind, carried out by internal and external reviewers of the publishing house. Articles are indexed in a variety of international scientific databases, and access to the journal's database is open to any reader. Publication in the journal takes place 4 times a year.

Publisher's website: https://www.professionalbulletinpublisher.com/

Issue № 3/2025
Brasov County, Romania

Научное издательство «Профессиональный вестник»

Журнал «Профессиональный вестник. Информационные технологии и безопасность»

Профессиональный вестник. Информационные технологии и безопасность – профессиональное научное издание. Публикация в нем рекомендована практикам и исследователям, которые стремятся найти решения для реальных задач и поделиться своим опытом с профессиональным сообществом. Публикация в журнале подходит для тех специалистов, кто работает и активно развивает передовые ИТ-решения, такие как технологии ИИ, блокчейна, больших данных и другие.

Журнал рецензирует все входящие материалы. Рецензирование – двойное слепое, осуществляется внутренними и внешними рецензентами издательства. Статьи индексируются во множестве международных научных баз, доступ к базе данных журнала открыт для любого читателя. Публикация журнала происходит 4 раза в год.

Сайт издательства: https://www.professionalbulletinpublisher.com/

Выпуск № 3/2025

Жудец Брашов, Румыния

Contents

Содержание выпуска

UDC 004.415.5: 004.738.5:339.3

THE IMPACT OF TESTING PRACTICES ON THE PERFORMANCE AND PROFITABILITY OF E-COMMERCE PLATFORMS AMID GROWING DIGITAL CONSUMPTION

Drogunova Y.
bachelor's degree, Dostoevsky Omsk state university (Omsk, Russia)

ВЛИЯНИЕ ПРАКТИК ТЕСТИРОВАНИЯ НА ПРОИЗВОДИТЕЛЬНОСТЬ И ДОХОДНОСТЬ E-COMMERCE-ПЛАТФОРМ В УСЛОВИЯХ РОСТА ЦИФРОВОГО ПОТРЕБЛЕНИЯ

Дрогунова Ю.И.
бакалавр, Омский государственный университет имени Ф. М. Достоевского (Омск, Россия)

Abstract

The article examines modern software testing practices in e-commerce and their influence on the performance and economic efficiency of digital platforms. It analyzes the role of quality assurance integration into CI/CD workflows, the automation of user scenarios, and real-time monitoring. The study highlights that the maturity of the testing infrastructure directly affects key metrics such as time-to-market, system resilience under load, and ROI. The implementation of testing strategies under conditions of growing digital consumption contributes to incident reduction, conversion rate improvement, and customer retention. The article concludes that quality assurance should be viewed as a strategic asset of a digital platform.

Keywords: testing, e-commerce, DevOps, automation, performance, ROI, resilience.

Аннотация

В статье рассматриваются современные практики тестирования программного обеспечения в электронной коммерции и их влияние на производительность и экономическую эффективность цифровых платформ. Анализируется роль интеграции гарантий качества в процессы CI/CD, автоматизации пользовательских сценариев, а также мониторинга в режиме реального времени. Подчеркивается, что зрелость тестовой инфраструктуры напрямую влияет на такие метрики, как время вывода продукта на рынок, устойчивость систем при нагрузках и ROI. Использование стратегий тестирования в условиях роста цифрового потребления способствует снижению инцидентов, увеличению конверсии и удержанию клиентов. Сделан вывод о необходимости рассматривать гарантии качества как стратегический актив цифровой платформы.

Ключевые слова: тестирование, e-commerce, DevOps, автоматизация, производительность, ROI, устойчивость.

Introduction

Rapid growth in digital consumption in recent years significantly impacted the dynamics of e-commerce evolution. E-commerce websites have evolved from simple sales channels to sophisticated digital ecosystems, and any malfunction of one component can result in instant financial loss, reduced customer confidence, and a drop in major indicators like conversion rates, user retention, and repeat purchases. In a competitive economy, customer expectations are not just limited to prices and product offerings anymore – now they also encompass speedy page loading, interface stability, and error-free

user experiences. Since system performance and reliability have a great deal to do with how advanced the quality assurance (QA) processes are, the test function is now increasingly being recognized as a leading force for both digital stability and business achievement.

The aim of this article is to analyze the impact of modern software testing practices on the performance and profitability of e-commerce platforms under conditions of increasing load and shifting consumer behavior. Particular attention is given to how the integration of QA into DevOps workflows, automation of user scenario testing, and real-time monitoring contribute to enhanced reliability metrics, reduced incident recovery time, and an overall increase in return on investment (ROI) from digital solutions.

Main part. Causal link between software quality and business performance in e-commerce

In the field of e-commerce, software quality is not merely a technical attribute but an economically significant factor that directly influences a platform's financial performance. Errors in user-facing scenarios – such as failures during checkout, delays in page transitions, or inconsistencies in shopping cart behavior – disrupt the user journey, erode trust, and directly reduce the platform's ability to convert visitors into paying customers. Such defects, particularly when persistent, contribute to increased user churn, lower customer satisfaction metrics, and a decline in customer lifetime value (LTV). According to estimates by research agency Grand View Research, the global market for e-commerce software exceeded $7,5 billion in 2024 (fig. 1).

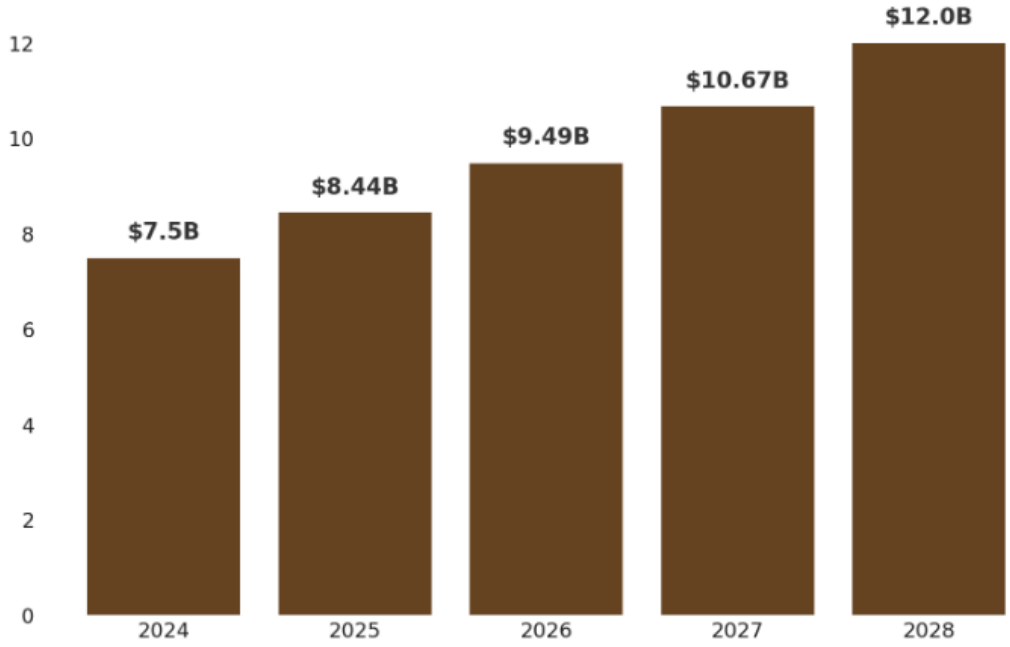


Figure 1. Projected global e-commerce software market size, 2024-2028 [1]

Notably, the adverse effects of poor quality are rarely immediate or transient. As, over time, technical debt accumulates through delayed refactoring or inadequate regression control, system stability is compromised and scalability is hampered. The pervasive degradation impacts velocity of feature releases and increases operating maintenance costs. At busy periods, say holiday seasons when sales are high, if technical debt is not cleared, it can manifest as high response times or even service shutdown (outage), both causing substantial revenue loss. Experts in the industry estimate that one minute of unplanned downtime for an e-commerce giant can translate to tens of thousands of lost sales.

It makes a direct cause-and-effect relationship between business performance and software quality. Poor quality creates system failures, which degrade user experience and trigger negative behavioral responses – reduced repeat business, higher support cost, and ultimately lower profitability. QA therefore needs to be viewed not only as a technical reliability control but as a strategic optimization control for important performance metrics (KPI) such as churn rate, customer satisfaction, and ROI in e-commerce.

Testing strategies for digital commerce

Developing effective QA methods for web stores includes paying close heed to platform architectural complexity, the over-sensitivity of user scenarios to failure, and the necessity of offering assurances of resilience under varying load conditions. Testing multiple levels – ranging from component-level validation to real-time monitoring – enable the identification of vulnerabilities at several stages of the system life cycle and assist in minimizing the effect of defects on core business metrics (table 1).

Table 1

Comparative overview of testing strategies in e-commerce [2, 3]

Testing strategy	Purpose	Method	Tools	Outcome
End-to-end testing of user scenarios	Validate business-critical user flows.	Simulate user actions.	Cypress, Playwright, Selenium.	Reduces failure risk during checkout and transaction processes.
Integration testing of microservices and API	Identify errors in inter-service communication.	API requests and contract validation.	Postman, RestAssured, Pact.	Ensures stability across distributed components.
Performance testing	Assess system resilience under high load.	Load simulation and stress scenarios.	JMeter, Gatling, Locust.	Readiness for peak traffic periods.
Synthetic monitoring	Monitor availability and speed along predefined routes.	Automated tests at scheduled intervals.	Pingdom, Uptrends, New Relic Synthetics.	Proactive detection of service disruptions.
Real-user monitoring (RUM)	Analyze actual user behavior in real-time.	Client-side telemetry collection.	Google Analytics, Datadog RUM, New Relic.	UX optimization based on real-world usage data.

The intersection of diverse testing strategies enables e-commerce sites to address QA from multiple dimensions – functional correctness, stress testing, and on-the-fly user friendliness. Rather than relying on one method, effective QA frameworks integrate automated scenario testing, robust API tests, active infrastructure monitoring, and behavior-based analytics to ensure system resilience and end-user satisfaction. This multi-layered approach not only reduces the likelihood of catastrophic failure but also allows for continuous improvement in key business metrics such as conversion rate, retention, and revenue stability in today's increasingly difficult digital environment.

Organizational and technological integration of QA into DevOps and product workflows

There are new commerce websites in a very dynamic environment – frequent releases, chaotic changing of user requirements, and constant functional growth require QA practices strongly ingrained in engineering and business processes. It requires shifting away from mainstream waterfall testing towards a **Shift-left model** where quality control begins as early as the definition of requirements and architectural design stages [4]. One of the QA effectiveness factors most crucial is its integration within the CI/CD pipeline in a smooth fashion (fig. 2).

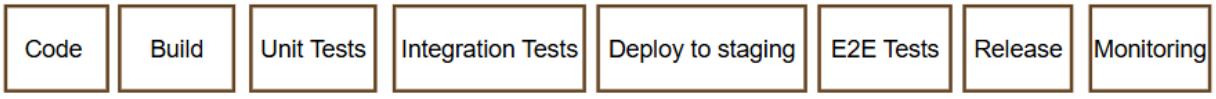


Figure 2. QA integration points in a typical CI/CD pipeline

Here, automated tests like unit, integration, and end-to-end tests are executed at every stage of the build and deployment lifecycle, providing instant feedback on artifact quality. Defects are identified and resolved early in the development cycle, and this reduces the overall cost of errors

appreciably. Continuous delivery is enabled by continuous monitoring of system performance and stability, i.e., Mean Time to Recovery (MTTR), which helps make judgments not only on the number of incidents but also response time and recovery.

More broadly, QA is increasingly evolving from an isolated technical function into a component of the product hypothesis itself – actively participating in the formulation and validation of business ideas [5]. Testing MVP (minimum viable products), conducting A/B experiments, and analyzing user feedback enable teams to make evidence-based decisions, minimizing the risk of misaligned priorities. In this context, quality is no longer viewed merely as a «post-development check»,but becomes a fundamental part of the digital product's value proposition. This logic is illustrated in figure 3, which presents the Dev–Test–Business Feedback Loop – showing the integration of hypothesis generation, development, testing, and real-time user feedback.

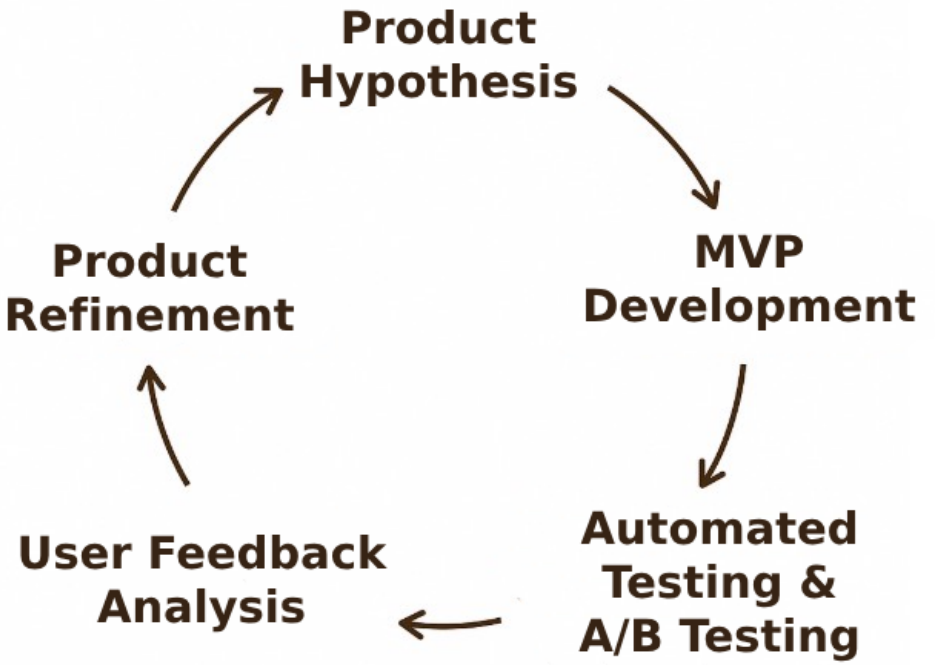


Figure 3. Dev – Test – Business feedback loop in product quality lifecycle

To assess the maturity of a QA infrastructure, two such critical measures are employed: Defect Leakage Rate and Mean Time to Recovery (MTTR). The former measures the rate of defects that escape detection during the test process and are subsequently discovered in production, serving as a measure of poor test coverage or inefficiency in the testing process. The latter measures the pace of returning a system to usual operation following a failure by a team, and thus indicates the operational resilience of the company. Combined examination of these metrics provides not only an estimate of the software quality per se, but also turns into a quantitative basis for the justification of investment in test automation and additional QA development to business stakeholders. In the context of increasing complexity and digital risk exposure in e-commerce systems, it is essential to align technical quality metrics with broader models of financial and operational risk management. The use of artificial intelligence in risk analysis offers practical approaches for anticipating failures and optimizing decision-making in digital environments [6].

Qualitative impact of software quality on e-commerce platform profitability

High software quality exerts a broad economic impact on the performance of e-commerce websites. It directly impacts – through diminished failure loss and improved conversion – and indirectly impacts – by creating greater customer trust, increased purchase frequency, and reduced support expenditures. In an economy with limited space for errors, investments in QA processes are not perceived as costs anymore but rather as profitability and ROI drivers [7].

From an economics perspective, the underlying effect of quality arises from reducing the incidence of transaction failure, particularly in mission-critical user flows such as registration, search, checkout, and payment. Even minor deficiency in these places can disrupt the journey of the user and lead to direct revenue loss. As per industry estimates, adding as little as 0,1% in availability for a site

with over 1 million daily hits can increase tens of thousands of dollars in additional monthly profits. Increased test coverage and automated testing deliver reliable system response under varying load, thereby rendering essential performance metrics such as conversion rate and AOV more stable.

One of the other noteworthy points is the UX optimization from QA-driven insights. RUM helps determine behavioral patterns and systemic problems across geographies, devices, and access channels. They are not only applied for defect resolution but also for determining product roadmap priorities, which have direct implications on customer retention and maximizing LTV.

Also noteworthy is the effect that adult QA practices have in reducing the cost of operations. Defect detection in the CI phase is 6-15 times less expensive than fixing the same errors at the time of production. Furthermore, an uptime platform minimizes the load on technical support teams, lowers SLA violations, and lowers escalation levels – resulting in a direct effect on improved business margins. Indicators such as Defect Containment Effectiveness (DCE) and Cost of Quality (CoQ) provide for a quantitative assessment of the effectiveness of QA investment in terms of financial results.

Also QA contributes to profitability by accelerating time to market. Automated regression and functional testing shorten release cycles, allowing for faster hypothesis validation and product iteration. This is particularly critical for platforms relying on dynamic pricing, personalized marketing, and rapid feature experimentation.

In summary, high-quality software is not merely a technical attribute – it is a strategic asset for any e-commerce platform. The maturity of QA processes directly determines a business's capacity to adapt, scale, reduce losses, and generate sustainable revenue. Embedding quality metrics into business analytics and executive reporting is becoming an essential practice for managing digital product profitability.

Comparative impact of testing practices on key metrics of e-commerce platforms

Against the backdrop of the accelerating rhythm of digital consumption and rising user expectations, e-commerce platforms are compelled to adopt diverse testing routines in order to deliver technical robustness and business competitiveness. They vary in efficiency depending on the maturity level, integration into product workflows, and suitability for the type of platform (e.g., B2C vs. B2B, omnichannel complexity).

To support strategic planning of QA infrastructure, the impact of some testing practices on performance and business metrics should be evaluated. The table 2 presents the comparative evaluation of the impact of various testing strategies on three main dimensions: time-to-market (TTM), system stability, and business performance (conversion rate, ROI, and other related metrics).

Table 2

Enhanced technical comparison of testing practices in e-commerce [8, 9]

Testing practice	**Time-to-market impact**	**System stability effect**	**Conversion / ROI influence**	**Cost efficiency / scalability**
Manual regression testing	Low (1–2 releases/month).	Moderate (limited depth, human error-prone).	Low (few business insights, limited scalability).	Low (labor-intensive, hard to scale).
Automated E2E testing	Medium (weekly deployments).	High (broad coverage of key user journeys).	Medium (improves trust and flow continuity).	Medium-High (requires setup, pays off at scale).
CI/CD pipeline QA integration	High (daily or continuous delivery).	Very High (real-time validation, shift-left model).	High (fewer rollbacks, better user stability).	Very High (automated, scalable, developer-aligned).

Testing practice	Time-to-market impact	System stability effect	Conversion / ROI influence	Cost efficiency / scalability
Performance / load testing	Indirect (affects stability, not speed).	Very High (prevents crashes at scale).	Medium (better UX during peak events).	High (valuable for large-scale or seasonal platforms).
Production monitoring (RUM + Synthetic)	None (post-release diagnostics only).	Moderate (detects live issues, no prevention).	High (exposes real-world bottlenecks).	Medium (scales with user base, analytics-dependent).
A/B testing and experimentation	Medium (depends on iteration speed).	Low (not focused on infrastructure faults).	Very High (data-driven UX and revenue optimization).	Medium (ROI tied to analytics and product maturity).

The evidence suggests that most strategically important practices are those that are built into continuous delivery and feedback cycles – i.e., CI/CD test automation, end-to-end automation, and A/B experimentation. These approaches shorten development cycles with improved system predictability and driving quantifiable increases in user behavior and revenue results at the same time.

Conversely, stand-alone performance testing and manual testing are viable in resource-constrained or legacy systems but do not scale well with increasing release frequency and traffic. Production monitoring, although not strictly a development tool, offers real-world problem detection in a timely fashion and enables retention through the ability to mitigate UX problems more quickly.

Lastly, testing strategies for e-commerce will need to be tailored to the platform's operating model, level of maturity, and its resource constraints. This comparison model can be used as a QA investment prioritization tool and as a basis for leveraging engineering and business teams' participation in shared performance and profitability goals.

Conclusion

The research determines that long-standing and systematically incorporated testing methods exert both direct and indirect influences on the functionality and profitability of e-commerce websites under pressures of heightened online consumption. Implementation of automated testing, adding QA to CI/CD pipelines, and monitoring and experimentation practices (e.g., A/B testing, RUM) result in more technical resilience, reduced time-to-market, and improved core business metrics such as conversion rate, average order value, and customer retention. QA becomes a strategic component of digital risk management in high-speed release environments with varying load.

Moreover, the study finds that effectiveness of capital and operating expenses varies significantly based on testing practices chosen. Fragmented or manual practices are low in scalability and bad at relating with financial performance whereas automated and metrics-based ones allow higher ROI and SLA incidents reduction. Thus, QA must be regarded as not only a technological shield, but also as an investment tool of immediate value to the financial sustainability of online platforms. Valuing QA maturity accordingly into digital planning is thus a critical factor for maintaining long-term competitiveness within e-commerce.

References

1. E-commerce Software Market Size & Trends / Grand View Research // URL: https://www.grandviewresearch.com/industry-analysis/e-commerce-software-market (data of application: 13.08.2025).

2. Abhilash V., Venkat S.H., Nishal S., Rajagopal S.M., Panda N. E-commerce evolution: Unleashing the potential of serverless microservices // 2024 15th International Conference on Computing Communication and Networking Technologies (ICCCNT). IEEE. 2024. P. 1-8.

3. Sidorov D., Kuznetcov I., Dudak A. Asynchronous programming for improving web application performance // ISJ Theoretical & Applied Science. 2024. Vol. 138. № 10. P. 197-201.

4. Lan Q., Kaul A., Pattanaik N.K.D., Pattanayak P., Pandurangan V. Securing Applications of Large Language Models: A Shift-Left Approach // 2024 IEEE International Conference on Electro Information Technology (eIT). IEEE. 2024. P. 1-2.

5. Safarli N.Z. Artificial intelligence in financial risk analysis: theory and practice // Professional Bulletin: Economics and Management. 2025. № 1/2025. P. 46-53.

6. Mirjat N.A. Quality Assurance in Devops Environments: Strategies, Tools, And Best Practices // Multidisciplinary Science Journal. 2024.Vol. 1. № 01. P. 60-65.

7. Jin L., Chen L. Exploring the impact of computer applications on cross-border e-commerce performance // IEEE Access. 2024. Vol. 12. P. 74861-74871.

8. Larsen N., Stallrich J., Sengupta S., Deng A., Kohavi R., Stevens N.T. Statistical challenges in online controlled experiments: A review of a/b testing methodology // The American Statistician. 2024. Vol. 78. № 2. P. 135-149.

9. Bolgov S. Automation of business processes using integration platforms and backend technologies // International Research Journal of Modernization in Engineering Technology and Science. 2024. Vol. 6(12). P. 3847-3851.

UDC 004.4

OPTIMIZATION OF FRONTEND APPLICATION PERFORMANCE: MODERN TECHNIQUES AND TOOLS

Garifullin R.
bachelor's degree, Saint Petersburg electrotechnical university «LETI» (Saint Petersburg, Russia)

ОПТИМИЗАЦИЯ ПРОИЗВОДИТЕЛЬНОСТИ ФРОНТЕНД-ПРИЛОЖЕНИЙ: СОВРЕМЕННЫЕ ТЕХНИКИ И ИНСТРУМЕНТЫ

Гарифуллин Р.Ш.
бакалавр, Санкт-Петербургский государственный электротехнический университет «ЛЭТИ» им. В. И. Ульянова (Ленина) (Санкт-Петербург, Россия)

Abstract

This study examines how modern performance optimization techniques influence the efficiency of frontend applications. Key techniques such as code splitting, lazy loading, tree shaking, resources minimization and compression, use of modern bundlers (Webpack, Vite), and use of Content Delivery Networks (CDN) are examined. The influence of these methods on key performance metrics is analyzed. Additionally, research and real-world case studies are considered to illustrate the effects of both optimization strategies and loading delays on user behaviour, engagement levels, and business indicators.

Keywords: performance optimization, frontend applications, code splitting, lazy loading, tree shaking, Content Delivery Network (CDN).

Аннотация

В данной статье исследуется влияние современных методов оптимизации производительности на эффективность работы фронтенд-приложений. Рассматриваются ключевые техники, такие как разделение кода, ленивая загрузка, метод устранения мертвого кода, минимизация и сжатие ресурсов, использование современных сборщиков (Webpack, Vite) и внедрение инфраструктуры Content Delivery Network (CDN). Изучается роль этих методов на основные метрики производительности. Рассматриваются исследования и практические примеры, демонстрирующие влияние как стратегий оптимизации, так и задержек в загрузке на поведение пользователей, уровень вовлеченности и бизнес-показатели.

Ключевые слова: оптимизация производительности, фронтенд-приложения, разделение кода, ленивая загрузка, метод устранения мертвого кода, Content Delivery Network (CDN).

Introduction

Complexity of web applications has grown exponentially, along with their resource demands. Fast load times and usability are expected by users, and search engines like Google include performance measures in site rank algorithms. Frontend application performance is most important to user experience, conversion, and a whole host of key performance measures; therefore, performance must be optimized.

Recent advancements in frontend tooling and methodology have enabled sophisticated techniques in load performance optimization, data transfer minimization, and interface responsiveness. Code splitting, lazy loading, tree shaking, and the use of modern-day bundlers such

as Webpack and Vite have been the most successful techniques in practice. With proper use, such techniques have high potential for loading duration reductions, consumption minimization, and enhancing web application usability. This purpose of this article is to examine contemporary methods for optimizing frontend performance.

Main part. Theoretical foundations of frontend performance optimization

Web application performance is a good indicator of their success and has direct implications for user experience, engagement, and achievement. Taming load time facilitates seamless interaction, whereas latency invites frustration and diminished retention. As more users anticipate instant access to digital content, frontend performance optimization has become essential to maintain competitiveness and achieve high usability [1].

Poor performance also affects search engine optimization (SEO). Google introduced **Core Web Vitals** in 2021, which is a set of metrics to use when measuring the user experience, with a strong emphasis on loading speed. Faster web apps rank better in search, gaining more organic traffic and visibility. Moreover, slow interfaces negatively affect the accessibility of web applications. Users with slow internet connections or outdated devices encounter difficulties when loading heavy pages, making applications less inclusive. This issue is relevant for global services operating in regions with unstable internet connectivity.

To objectively assess web application performance, key metrics have been developed to measure various aspects of user experience (table 1).

Table 1

Web performance metrics [2, 3]

Metric	Description	Recommended threshold
Time to First Byte (TTFB)	Time taken for the browser to receive the first byte of the response from the server.	< 200 ms
First Contentful Paint (FCP)	Time taken to render the first visible element (text, image) on the screen.	< 1.8 s
Largest Contentful Paint (LCP)	Time taken to load the largest visible content (image, text block) on the page.	< 2.5 s
Cumulative Layout Shift (CLS)	Measures visual stability by calculating unexpected layout shifts during page load.	< 0.1
Time to Interactive (TTI)	Time taken until the page becomes fully interactive (when users can interact without delay).	< 5 s
Interaction to Next Paint (INP)	Evaluates the delay between a user interaction (click, keypress) and the next visible update on the page.	< 200 ms

Optimized websites offer real-time loading and smooth interface performance, which reduce the bounce rate and improve session duration. Users expect a site to load in three seconds, and each additional second of lag significantly increases the page abandonment rate. A Google research found that slowing down the load time of mobile pages from one to five seconds boosts the likelihood of bounce by 90% (fig. 1).

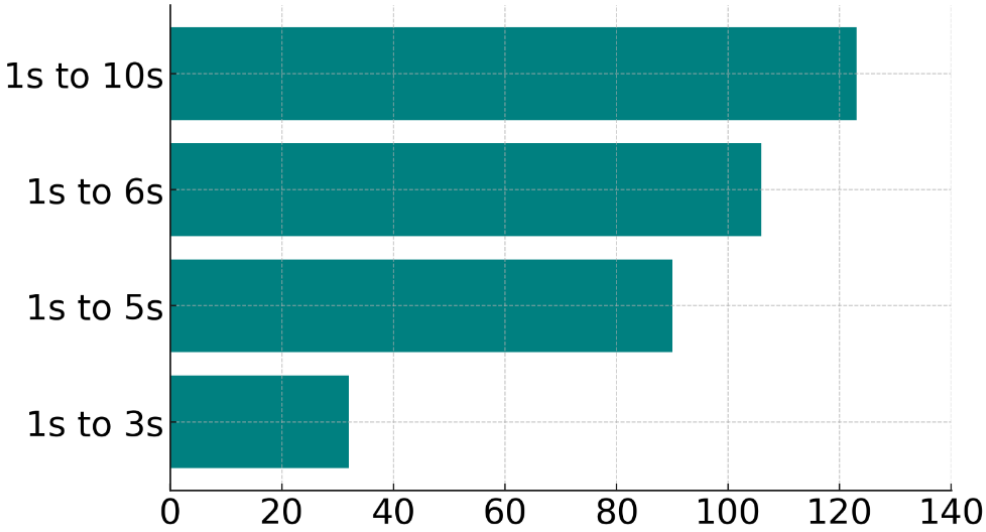


Figure 1. Increase in bounce probability based on page load time, % [4]

Performance is also a key factor in **mobile applications** and **Progressive Web Apps (PWA)**. A significant proportion of visitors access websites through mobile devices, many of which have recurring low network bandwidths compared with desktops. Optimising web applications will become increasingly important for enhancing smartphone usability, particularly in high volumes of mobile usage regions.

There have been a variety of specialist tools developed to evaluate and tune web application performance, with evaluation facilitated through automation and performance constraint identification. **Google Lighthouse** is a test tool that audits the performance, SEO, and accessibility of web applications and gives detailed optimization recommendations. **WebPageTest** is a tester that conducts page load speed tests in real-world scenarios on various devices, network types, and geographies. **Chrome DevTools** is a group of developer tools that come pre-installed with the Chrome browser and allow one to analyze performance metrics, resource loading, and blocking operations. Core Web Vitals Report, in Google Search Console, provides one with a view of Core Web Vitals metrics of website pages. PageSpeed Insights is another tool at Google that tests page speed and gives suggestions to enhance it.

Using the above tools, developers are able to perform unbiassed frontend application performance assessments and make evidence-based optimisation decisions. Regular measurement of key performance indicators identifies performance hotspots, optimises optimisation methods based on real user profiles, and provides consistent application response under increasing traffic and computational demands.

Modern techniques and tools for optimization

With the growing complexity of web applications comes a larger volume of data to be transferred, which has an unfavorable impact on loading time and performance. Heavy JavaScript bundles, resource-intensive resources, and redundant code only add to the load carried by the browser, further degrading the user experience. To address these issues, developers employ all manner of optimizations intended to reduce page load latency, remove unnecessary memory usage, and enhance interface responsiveness [5].

One of the most critical frontend application optimization techniques is code splitting, which allows a portion of an application to be loaded only when needed at a particular time. This minimizes data transferred and page loads. Code splitting is done through dynamic imports (import()), and thus modules are loaded on demand. Well-known frameworks such as React, Vue, and Angular have native support for dynamic loading. Dynamic loading allows the browser to load components or pages on demand rather than loading the entire application code at boot. Code splitting comes in handy for Multi-Page Applications (MPA) and Single-Page Applications (SPA) with massive amounts of functionality where a significant amount of functionality may not be utilized during the first load.

Lazy loading is implemented for code and also for media resources such as images and videos to reduce the first-load time and enhance the performance overall. Lazy loading would only load non-critical resources when they are needed, thereby lessening the data loaded at page startup, which is extremely beneficial for media-rich sites. In HTML5, deferred image loading could be implemented through the use of the attribute loading="lazy":

<img src="image.jpg" alt="Example" loading="lazy">

Lazy loading significantly reduces the initial data transferred and improves key performance metrics such as LCP [6]. For videos and frames, lazy loading is possible by introducing the loading="lazy" attribute or implementing JavaScript-based intersection observers to control when resources are loaded. This technique can be especially handy for long-scrolling pages, news websites, blogs, and e-commerce websites, where numerous images or videos may not be visible right after page load.

Tree shaking is a method that eliminates dead code at build time. The method works exceptionally well with ES6 modules (ECMAScript 2015) since they allow static dependency analysis, which makes it easy for bundlers to identify and remove unused exports. Tree shaking is natively supported by new build tools such as Webpack, Rollup, and Esbuild. In Webpack, for example, tree shaking is automatically enabled in production mode but turned off in development mode for the sake of debugging. Also, tree shaking can be further improved by configuring the sideEffects field in the package.json file so that files with no side effects are tree shaken in an optimal way. By removing dead code, this mechanism significantly reduces the final bundle size, which enhances loading performance and speed, especially in projects that have enormous external libraries.

A **Content Delivery Network (CDN)** is a server network geographically dispersed with the aim of serving static files from closer geographic locations to users in a bid to reduce latency and enhance web application reliability. A cache and distribution of assets such as images, stylesheets, JavaScript files, and even entire HTML pages at multiple edge locations reduces content fetch time significantly. This approach minimizes reliance on a single origin server, sharing traffic loads and increasing tolerance towards sudden peaks in high traffic or DDoS attacks. Additionally, most modern CDNs are supplemented with compression techniques including Brotli and Gzip, adaptive image compression, and HTTP/3, minimizing load time even more.

Minification and file compression make data transmitted smaller, thus faster page loading speed and performance. JavaScript and CSS minification is elimination of unwanted whitespace, comments, and unused code, which is easily accomplished with plugins such as Terser for JavaScript and CSSNano for CSS. Besides shortening the load of the script, the minification decreases the payload overall, which enhances the frontend performance and server response time. Gzip and Brotli compression algorithms also improve performance through HTTP response size compression prior to transmission to the client. Brotli provides higher compression ratios for text content like HTML, CSS, and JavaScript, with bandwidth efficiency being significantly increased.

Frontend performance is one of the strongest drivers of user experience, engagement, and business success for web applications. With first-order performance metrics knowledge and a set of specialized analysis tools at their disposal, developers can monitor bottlenecks and tune specifically. Through continuous monitoring and optimization of frontend performance, companies can improve load speed, accessibility, and competitiveness in a more demanding online marketplace.

Analysis of performance optimization and its impact on user engagement in companies

Performance optimization is a critical element in user action and business achievement. World case studies and research findings illustrate how anticipatory optimizations and performance lags affect user behavior, conversion rates, and system performance.

Netflix conducted a performance optimization initiative that was focused on making the unauthenticated user loading experience better [7]. The primary aim was to optimize TTI, which is an essential metric that tracks how long it takes a page to be interactive. For this purpose, the development team implemented prerendering and critical rendering path optimization. These improvements resulted in faster initial content presentation as well as generally improved perceived loading speed.

Research by **Google** further confirms that even minor delays in page loading can significantly impact user behaviour [8]. In an experiment, the company actually delayed search result loading times by 100-400 milliseconds to see how user behavior altered. The results were that 100 milliseconds of delay dropped search behavior by 0,2% during the first six weeks and 400 milliseconds dropped search behavior by 0,6%.

Besides, the adverse impact became more potent over time: users who were repeatedly faced with prolonged page loads kept reducing use over time, and even after performance was restored, they failed to return to levels of engagement before. Surprisingly, after six weeks of exposure to a 400-millisecond delay, users kept conducting 0,21% fewer searches during the following five weeks after the experiment ended.

A **2020 Deloitte study** further underscores the significance of load speed optimization [9]. It demonstrated that improving mobile page loads by just 0,1 seconds had a return of 8,4% increase in conversion rate in retail and 10,1% in travel. Mean order value also increased by 9,2% in retail and 1,9% in travel, showing the real financial benefit of performance optimization. These findings confirm the key role of frontend performance improvement to efficiency, user engagement, and overall profitability in web applications.

Conclusion

Optimization of frontend application performance is a crucial aspect of rendering a user's experience fast, business metrics improved, and the overall performance of web applications. Emerging mechanisms such as code splitting, tree shaking, lazy loading, and employing advanced bundlers and CDN greatly reduce page load times and render the interface more responsive. Empirical evidence from top tech companies shows that performance investments carry a significant weight regarding high conversion rates, low bounce rates, and overall improvement in terms of user experiences. As web app usability and speed are in high demand, developers committed to offering high-quality digital products must ensure continuous improvements in terms of optimization techniques.

References

1. Sidorov D., Kuznetcov I., Dudak A. Asynchronous programming for improving web application performance // ISJ Theoretical & Applied Science. 2024. Vol. 138. № 10. P. 197-201.
2. Wehner N., Amir M., Seufert M., Schatz R., A vital improvement? Relating Google's core web vitals to actual web QoE // 2022 14th international conference on quality of multimedia experience (QoMEX). IEEE. 2022. P. 1-6.
3. Dobbala M.K., Lingolu M.S.S. Web Performance Tooling and the Importance of Web Vitals // Journal of Technological Innovations. 2022. Vol. 3. № 3.
4. Find Out How You Stack Up to New Industry Benchmarks for Mobile Page Speed / GoogleAPIs / URL: https://think.storage.googleapis.com/docs/mobile-page-speed-new-industry-benchmarks.pdf (date of application: 24.08.2025).
5. Dudak A. Object-oriented design patterns in front-end development // International independent scientific journal. 2024. № 66. P. 67-70.
6. Bâra R.M., Boiangiu C.A., Tudose C. Analysing the performance impacts of lazy loading in web applications // Journal of Information Systems & Operations Management. 2024. Vol. 18. № 1. P. 1-15.
7. Ahmed S., Aziz N.A. Impact of ai on customer experience in video streaming services: A focus on personalization and trust // International Journal of Human. Computer Interaction. 2024. P. 1-20.
8. Speed Matters / Google Research // URL: https://research.google/blog/speed-matters/ (date of application: 24.08.2025).
9. Milliseconds Make Millions / Deloitte // URL: https://www.deloitte.com/ie/en/services/consulting/research/milliseconds-make-millions.html (date of application: 24.08.2025).

UDC 004.4

COMPARATIVE ANALYSIS OF PERFORMANCE AND SCALABILITY OF SYNCHRONOUS AND ASYNCHRONOUS INTERACTIONS IN MICROSERVICE ARCHITECTURE

Smirnov A.
master's degree, Perm national research polytechnic university
(Perm, Russia)

СРАВНИТЕЛЬНЫЙ АНАЛИЗ ПРОИЗВОДИТЕЛЬНОСТИ И МАСШТАБИРУЕМОСТИ СИНХРОННЫХ И АСИНХРОННЫХ ВЗАИМОДЕЙСТВИЙ В МИКРОСЕРВИСНОЙ АРХИТЕКТУРЕ

Смирнов А.В.
магистр, Пермский национальный исследовательский политехнический университет (Пермь, Россия)

Abstract

This paper explores the choice between synchronous and asynchronous interaction models in microservice architecture, focusing on their impact on system performance and scalability. Key characteristics such as latency, throughput, and scalability under varying loads are discussed. The interaction models based on REST API and event-driven approaches are compared, emphasizing their advantages and disadvantages in the context of high-load distributed systems. The analysis presented aids in selecting the optimal interaction model based on system-specific requirements, such as data consistency, fault tolerance, and processing efficiency.

Keywords: microservice architecture, synchronous interactions, asynchronous interactions, performance, scalability, REST API.

Аннотация

В статье проводится сравнение между синхронными и асинхронными моделями взаимодействия в микросервисной архитектуре, акцентируется внимание на их влиянии на производительность и масштабируемость систем. Описываются ключевые характеристики этих подходов, такие как механизмы задержек, пропускная способность и масштабируемость при различных нагрузках. Сравниваются модели взаимодействия, основанные на REST API и event-driven подходах, с акцентом на их преимущества и недостатки в контексте высоконагруженных распределенных систем. Представленный анализ помогает в выборе оптимальной модели взаимодействия в зависимости от специфики требований к системе, таких как согласованность данных, отказоустойчивость и эффективность обработки.

Ключевые слова: микросервисная архитектура, синхронные взаимодействия, асинхронные взаимодействия, производительность, масштабируемость, REST API.

Introduction

New information systems, which are created on the basis of the microservice architecture, must select the most suitable mechanism of interaction between services. There are various versions of interaction: synchronous (for example, REST) and asynchronous (according to the event exchange principle). They are both pros and cons in terms of system performance, scalability, data consistency, and load tolerance. One of the most significant issues is how to balance the trade between

predictability and strong consistency on synchronous calls against flexibility at potentially higher complexity using asynchronous interaction.

The relevance of this research stems from rising needs for dealing with large amounts of information and high availability in today's distributed systems. The proliferation of applications with microservices orientation calls for the understanding of how diverse interaction models affect delay on messages upon sending, data consistency, and resource utilization.

The purpose of this research is to make a comparative analysis of scalability and performance of two interaction models of microservices. For these purposes, theoretical ideas of their work are used, primary performance indicators are analyzed, and the impact of choosing an interaction model on data consistency is researched. The research is carried out with the help of analysis of available scientific literature, study of theoretical interaction models and their assessment based on most significant characteristics.

Main part. Theoretical basis of interactions in microservice architecture

Microservice architecture is a software development style where a system is broken down into tiny independent services that exchange information with one another using network protocols. The interaction mechanism between services significantly influences the system's performance, data consistency, and scalability. These come in two broad forms: synchronous interaction, involving a direct request and response, and asynchronous interaction, involving sending and receiving messages or events and no feedback being provided immediately.

The synchronous approach, predominantly used via the REST API (Representational State Transfer), offers tight coupling among services. In the process, the client sends a request to the server and then waits for the reply before further moving ahead to execute its logic. The mechanism is simple to implement in deterministic processes as it offers predictability to operations and simplifies error management (fig. 1).

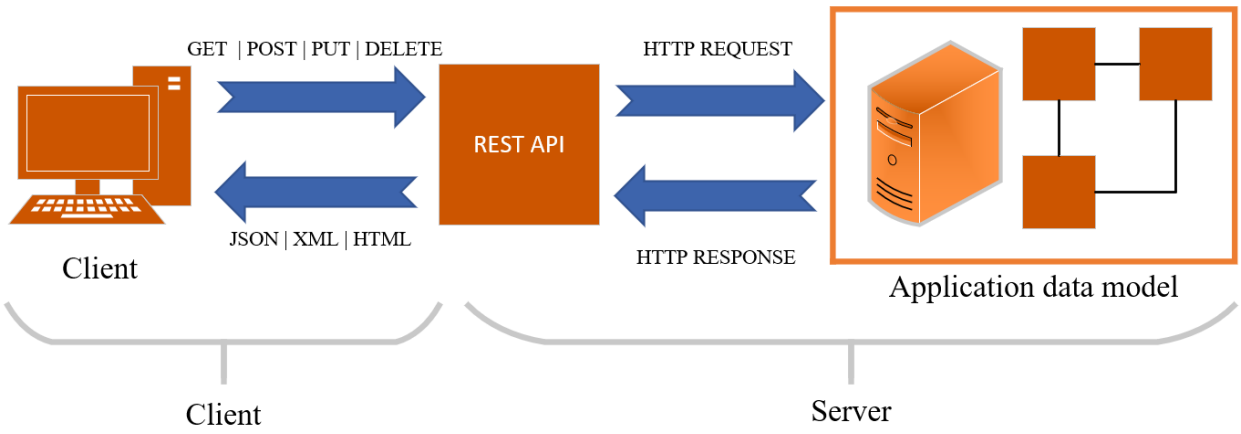


Figure 1. REST API framework [1]

However, extensive reliance among services creates greater delays as the weight increases. The greater the number of calls placed, the higher the number of REST requests that will form «cascades of delays» since every request will continue to hinder operation until it is received. Bottlenecks and performance are realized under high traffic densities in a system.

The asynchronous approach, however, relies on sending messages between services without the anticipation of a prompt response. This is achieved using message brokers such as Apache Kafka, RabbitMQ, or AWS SQS. In asynchronous systems, the sender sends the message to a queue where it is picked up by the receiver for processing. This method removes the dependency of services on each other and makes them fault tolerant, as services will keep functioning regardless of whatever state other parts are in (fig. 2).

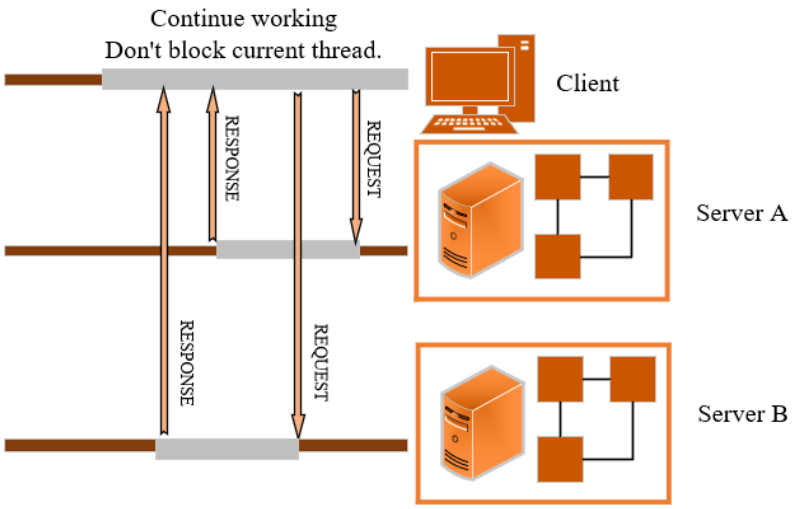


Figure 2. Asynchronous approach framework [2]

However, this approach makes it difficult to ensure strict consistency of data, as the operation can stop at a random point. Distributed systems employ eventual consistency approaches where consistency is achieved through replication and repeated execution of operations on failure.

Another factor while choosing an interaction model is the scalability issue. Horizontal scaling with additional service replicas and load balancing in synchronous systems means additional management costs of extra infrastructure. However, in asynchronous systems, by virtue of the message queues, the load gets dynamically distributed with flexible reclamation of resources across nodes [3].

Thus, synchronous or asynchronous interaction is selected based on the type of system and its performance requirements. Synchronous is appropriate for transactional operations with high consistency requirements, while the asynchronous method is better suited for highly loaded distributed systems where reducing interdependence between services is important.

Performance and scalability metrics

Measuring the efficiency of interactions in a microservice system is based on analyzing key indicators that reflect query processing time, the use of computing capacities, and the system's adaptability to changing workloads. Among them, **response time**, **bandwidth**, and **infrastructure costs** are particularly significant. The choice between synchronous and asynchronous interaction significantly affects these parameters, which is why it is necessary to study them in detail.

One of the more important performance characteristics is response time, or latency, which measures how long it takes from the time a request is sent to the time a response is received. Response time in synchronous systems is directly proportional to the number of services involved in the call chain. For example, in sequential REST communication, where one request takes 100 ms and five services must complete the operation, the overall delay can reach 500 ms, without taking network overhead into account [4]. In asynchronous systems, delays can be distributed over time, reducing the load on individual components, but in scenarios with strict consistency requirements, this can lead to an increase in overall latency.

Another important metric is throughput – the number of requests processed in a time unit. Knowledge of this metric in synchronous systems is limited by server capacity and the number of simultaneous connections, while in asynchronous architectures it can be enhanced by scalable message queues. For example, the use of Apache Kafka allows processing tens of thousands of messages per second with effective load balancing [5]. However, high throughput does not always imply low latency and requires careful consideration of usage scenarios.

Scalability is significantly affected by the way the computing resources are used. In synchronous architectures, each blocked thread consumes RAM and CPU cycles, which lead to resource exhaustion under high load. Asynchronous models use non-blocking processing and distributed queues to use resources more effectively, with less memory consumption under the same load. This is at the expense of extra state negotiation mechanisms and message relaying in case of

failures, which may make the system more complex. However, systems designed to scale usually have these mechanisms in place anyway.

Comparative analysis of REST and event-driven interactions

The selection of REST interaction or event-driven communication within a microservice architecture depends on differences in the request processing mechanisms, delay of data transfer, fault tolerance, and scalability. Each of them has its advantages and disadvantages that should be taken into account during high-load system design.

REST adopts a synchronous pattern of interaction, in which the client sends an HTTP request and then waits for the server's response. This is more convenient to deal with errors and maintain the system under control, as every call means real-time data processing. But REST in large distributed systems is plagued by «cascading dependencies», where one service downtime can lead to unavailability of a series of connected components. For example, if the auth service does not respond, it blocks access to all dependent services.

Unlike REST, event-driven systems utilize asynchronous message-based communication via brokers such as Apache Kafka or RabbitMQ. Here, services publish events that can be handled by one or more subscribers. This increases scalability and fault tolerance because services are not dependent on other services being ready immediately. The complexity of implementation is greater as coordination of the state between different components of the system needs to be done.

Also, in server and client-side applications, **RxJS** (Reactive Extensions for JavaScript) and **NgRx** (Angular Reactive Extensions) enable efficient data management and event flow, and thus they are useful tools in event-driven systems. RxJS enables you to create reactive data flows and manage asynchronous computation, and NgRx implements the Redux pattern in Angular applications so that state is easier to manage in an async environment. These tools demonstrate the reactive programming principles, being fault-tolerant and scalable without blocking the threads. They are especially useful with message-driven architecture, where effective system state management and real-time event processing are required [6].

From a performance perspective, REST-based systems can have less **latency with minimal services since** the response to the request is immediately available. With higher load, however, REST begins to suffer: an increase in the number of concurrent connections leads to blocking threads and response time. In event-driven architecture, the load is dynamically distributed, and asynchronous processing allows you to enhance the system throughput. For example, in research, transitioning to the event-driven model lowered the utilization of processor resources by 77,5% and lowered average response delay with large load [7].

The second distinction lies in data consistency mechanisms. Within REST-based systems, transactions are usually performed in real time with strict consistency. Event-driven systems usually apply the event consistency model, in which updates are broadcasted with a delay. This can cause temporary inconsistency in data between services, which is of critical importance for banking systems or online shopping websites. Thus, therefore, the REST style is still favored for tightly consistent and integrated systems, but event-driven systems are more scalable and fault-resistant under high loads.

Choosing the optimal approach depending on the usage scenario

The optimum choice between synchronous REST communication and asynchronous event-driven communication depends on the system's particularities, performance requirements, data consistency, and fault tolerance. In different situations, one of these approaches may be preferable, providing the best tradeoff between implementation ease and operational efficiency (table 1).

Table 1

Comparison of approaches based on use case scenarios

Scenario	REST (synchronous communication)	Event-Driven (asynchronous communication)
Transactional systems	Suitable for systems where strict data consistency and integrity are critical (e.g., banking applications).	Can be applied in scenarios where consistency can be temporarily compromised, but efficiency is still required.

Scenario	REST (synchronous communication)	Event-Driven (asynchronous communication)
Real-time processing	Not always optimal as high server load may lead to delays.	Suitable for real-time event processing with minimal latency (e.g., monitoring systems).
High load systems	Latency may accumulate with an increase in the number of requests. Scalability is limited.	Suitable for systems with high event volumes, such as IoT platforms, as it can handle parallel events efficiently.
Inter-service communication	Problems with cascading dependencies where the failure of one service blocks others.	Resilient to failures due to service independence and ability to process events independently.
Handling large data volumes	Not always efficient at handling large volumes of data due to scalability limitations.	Suitable for streaming data processing, such as real-time data streams, using message brokers (e.g., Kafka).
Flexibility and fault tolerance	May be less flexible in failure conditions as requests block execution until a response is received.	Provides high flexibility and fault tolerance, as services can continue processing messages regardless of others' states.

In those applications where data consistency is required to be strict, REST-style interactions remain the most reliable choice. For example, in banking applications or accounting software, it is crucial that every transaction is executed in a strictly defined order and does not provide any scope for inconsistencies. REST supports direct request processing, and ACID guarantees minimize the chances of incorrect operations. But with more requests, there is a scalability issue, as every service call threads-blocks until a response is received [8].

Conversely, **event-driven systems** are extremely effective in cases where a large number of events have to be processed with very low inter-service dependency. For instance, in processing systems for streaming data like IoT platforms or analytics services, the utilization of asynchronous message queues can drastically enhance throughput. In such systems, delays in processing an individual event are less significant than the ability for horizontal scaling and overload tolerance.

Failure tolerance is another critical feature. Using the REST model, failure of any one of the primary services can propagate through the system since the clients are expecting a synchronous response. This necessitates load balancers and retry logic, which is an increased level of infrastructure complexity. In event-driven systems, losing a service does not obstruct request processing, as events are stored in message brokers temporarily. However, such architectures require additional management of event handling and deduplication logic, especially if the data is critical.

Hence, the use of either REST or event-driven is determined by the requirements of a given scenario. When data consistency with low integration complexity is needed in a system, REST remains the best choice. When fault tolerance and scalability are crucial in heavy loads, event-driven interaction is more flexible and efficient in handling events. In some cases, hybrid models combining both approaches are used with a blend of both depending on operation urgency and latency requirements.

Conclusion

Comparative analysis of synchronous REST calls and asynchronous event-driven communications within microservice architecture has shown that the best method is determined by requirements in terms of performance, scalability, and data consistency. REST provides predictability as well as high consistency but is limited to blocking calls and therefore reduces its efficiency in cases of high loads. Event-driven designs allow you to achieve high throughput and fault tolerance but require additional mechanisms for state matching as well as control over event processing order. In scenarios that require low latency and deterministic execution of operations, REST remains among the best solutions. In high parallelism systems and dynamic load, the event-driven design works better. In some cases, the optimum is a hybrid model that combines the strengths of each to find balance in consistency, processing velocity, and scalability quickness.

References

1. Lercher A. Managing API Evolution in Microservice Architecture. // In Proceedings of the 2024 IEEE/ACM 46th International Conference on Software Engineering: Companion Proceedings. 2024. P. 195-197.

2. Söylemez M., Tekinerdogan B., Tarhan A. Microservice reference architecture design: A multi-case study. // Software: Practice and Experience. 2024. Vol. 54. № 1. P. 58-84.

3. Batista C., Morais F., Cavalcante E., Batista T., Proença B., Rodrigues Cavalcante W. Managing asynchronous workloads in a multi-tenant microservice enterprise environment. // Software: Practice and Experience. 2024. Vol. 54. № 2. P. 334-359.

4. Traini L., Cortellessa V. Delag: Using multi-objective optimization to enhance the detection of latency degradation patterns in service-based systems. // IEEE Transactions on Software Engineering. 2024. Vol. 49. № 6. P. 3554-3580.

5. Kafka 2.0 Documentation / Kafka // URL: https://kafka.apache.org/20/documentation.html (date of application: 18.08.2025).

6. Garifullin R. Application of RxJS and NgRx for reactive programming in industrial web development: methods for managing asynchronous data streams and application state // International Journal of Professional Science. 2024. № 12-2. P. 42-47.

7. WellRight modernizes to an event-driven architecture to manage bursty and unpredictable traffic / Amazon Web Services // URL: https://aws.amazon.com/ru/blogs/architecture/wellright-modernizes-to-an-event-driven-architecture-to-manage-bursty-and-unpredictable-traffic/ (date of application: 18.08.2025).

8. ACID Properties in DBMS / Geeks for Geeks // URL: https://www.geeksforgeeks.org/acid-properties-in-dbms/ (date of application: 18.08.2025).

UDC 004.415: 004.42

DYNAMIC TRAFFIC CONTROL MECHANISMS IN DISTRIBUTED SYSTEMS AS A MEANS OF ENSURING ADAPTABILITY AND FAULT TOLERANCE IN DIGITAL INFRASTRUCTURE

Terletska K.
bachelor's degree, Lviv polytechnic national university
(Lviv, Ukraine)

МЕХАНИЗМЫ ДИНАМИЧЕСКОГО УПРАВЛЕНИЯ ТРАФИКОМ В РАСПРЕДЕЛЕННЫХ СИСТЕМАХ КАК СРЕДСТВО ОБЕСПЕЧЕНИЯ АДАПТИВНОСТИ И ОТКАЗОУСТОЙЧИВОСТИ ЦИФРОВОЙ ИНФРАСТРУКТУРЫ

Терлецька Х.В.
бакалавр, Национальный университет «Львовская политехника»
(Львов, Украина)

Abstract

This article examines modern dynamic traffic control mechanisms in distributed computing systems as a means of enhancing the adaptability and fault tolerance of digital infrastructure. It analyzes load balancing algorithms, adaptive rate limiting, low-priority request shedding, and resource scaling strategies. Special attention is given to the integration of these algorithms into cloud-native architectures. The article emphasizes the role of observability as a foundational element enabling the transition from reactive to proactive traffic management. It concludes that the combined application of these mechanisms establishes a robust architectural foundation for the stable operation of distributed systems under high-load conditions.

Keywords: distributed systems, traffic management, scaling, fault tolerance, observability, cloud architectures, digital infrastructure.

Аннотация

В статье рассматриваются современные механизмы динамического управления трафиком в распределенных вычислительных системах как инструмент повышения адаптивности и отказоустойчивости цифровой инфраструктуры. Анализируются алгоритмы балансировки нагрузки, адаптивного ограничения скорости, сброса низкоприоритетных запросов и масштабирования ресурсов. Особое внимание уделяется интеграции алгоритмов в облачные архитектуры. Подчеркивается роль наблюдаемости как системообразующего элемента, позволяющего перейти от реактивного к проактивному управлению трафиком. Делается вывод о том, что совокупность этих механизмов формирует надежную архитектурную основу для устойчивого функционирования распределенных систем в условиях высокой нагрузки.

Ключевые слова: распределенные системы, управление трафиком, масштабирование, отказоустойчивость, наблюдаемость, облачные архитектуры, цифровая инфраструктура.

Introduction

Distributed computing system forms the foundation of digital infrastructure that offers scalability, availability, and continuity of service under constantly varying network loads. Increasing growth in data volumes causes traffic variability to increase in significance, where abrupt peaks and

irregular distribution of requests are typical. This necessitates the introduction of adaptive traffic control mechanisms that can appropriately react to external and internal variations in real time without impacting system performance or reliability.

Conventional static resource allocation techniques have proved ineffective under highly dynamic load situations, and hence the need for intelligent and adaptive mechanisms. Load balancing algorithms, adaptive request throttling, low-priority request shedding, and predictive scaling based on behavioral traffic analysis are gaining particular importance as key techniques in this context.

The goal of this study is to analyze and systematize dynamic traffic management mechanisms in distributed systems, with a focus on enhancing adaptability and fault tolerance. The research covers both the algorithmic principles of these mechanisms and their practical implementation in cloud-native environments, including Kubernetes, Istio, Envoy, and other components of modern digital platforms.

Main part. Traffic management in distributed architectures

Traffic control in distributed computing systems is a service whose impact is directly felt on the stability and performance of the entire digital infrastructure. The most common reason for instability is the **sudden spikes in load** that occur due to seasonal, day-by-day, or event-driven fluctuations in user traffic. Flash sales, live streaming, and viral social network clips are typical examples that can produce an immediate flood of incoming requests.

Another critical parameter is **request asymmetry** – a state of affairs whereby certain components of the system is disproportionately heavily loaded compared to other components. This disparity can be due to heterogeneous user patterns, non-uniform data distribution, or application-level semantics. Added on top of this is the **distributed nature of modern designs**: geo-distributed nodes, microservice-based deployments, and independently scaling components imply that request handling coordination has to withstand and adjust to adaptive routing schemes.

In such decentralized environments, ensuring trusted coordination and authentication between distributed components becomes increasingly important. Decentralized authentication methods improve system resilience by reducing dependency on centralized control points and enabling autonomous trust verification within distributed network topologies [1].

In response to these challenges, distributed systems actively employ load **balancing mechanisms** that ensure the even distribution of incoming requests across available service instances or nodes (fig. 1).

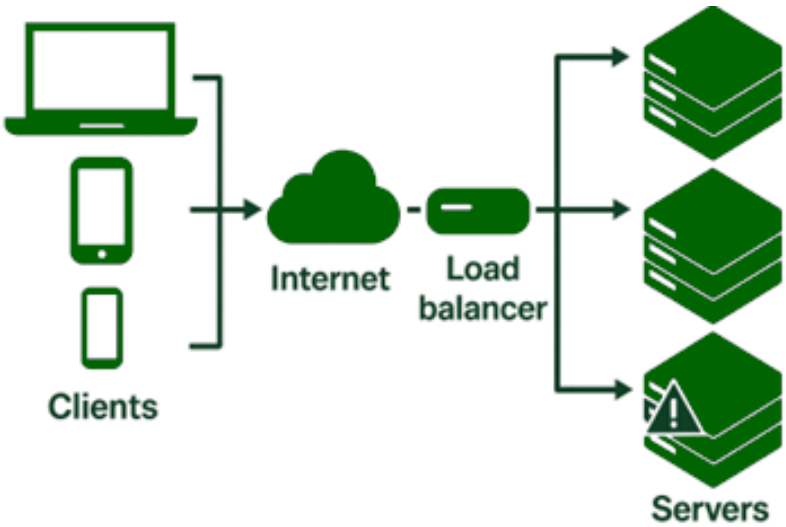


Figure 1. Load balancing architecture

Load balancing may also be done at the network layer (L4) as well as application layer (L7), depending on criteria such as request path, headers, client IP, and target service's workload. Existing deployment such as Envoy, NGINX, and HAProxy already support the use of algorithms such as round-robin, least connections, random, and user-supplied custom routing based on user-specified values. Even distribution evenness is also a concern in dynamically scalable systems but not the sole

consideration; latency, geographical node proximity, and network anomaly resilience also must be taken into account.

Alongside load balancing, adaptive throttling mechanisms are increasingly being **adopted to control service** throughput under overload conditions (fig. 2).

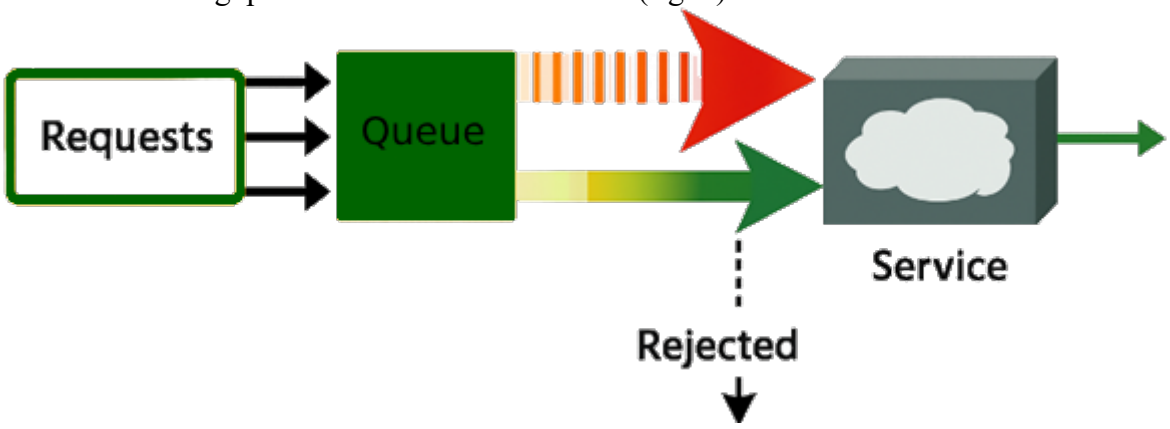


Figure 2. Adaptive throttling mechanism

These systems change the volume of incoming requests adaptively based on current load, response latency, or errors being experienced. As compared to fixed rate limiting, the adaptive version provides more dynamic utilization of resources and reduces the likelihood of failure in the event of brief traffic surges.

A complementary technique is **request shaping,** which involves the preliminary classification and prioritization of incoming requests (fig. 3).

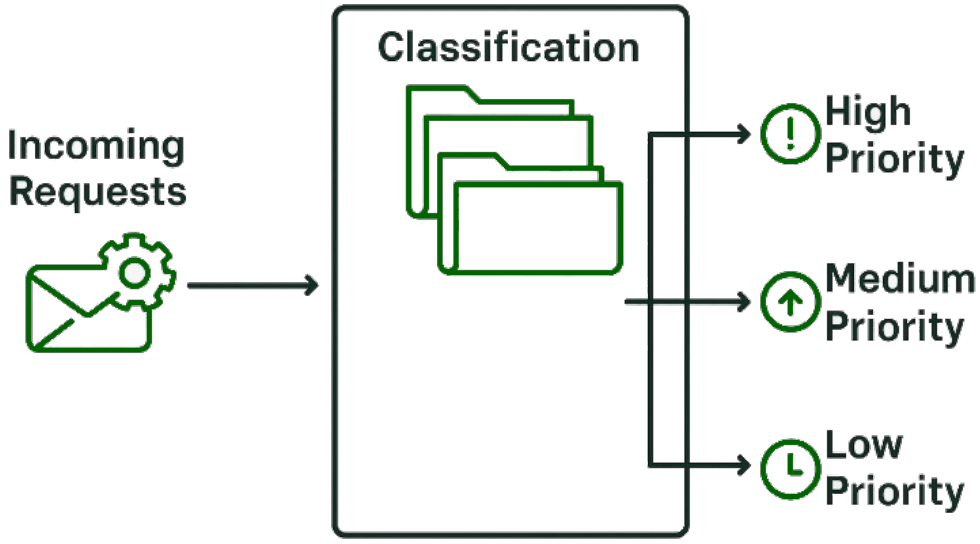


Figure 3. Request shaping architecture

For example, lower-priority tasks such as background sync or analytics requests may be delayed or batched, and higher-priority operations would be done as fast as possible. This approach improves worldwide service quality and avoids system measurements from exceeding operational limits even during constrained computational resources.

Thus, effective traffic management in distributed systems requires the joint usage of load balancing, adaptive throttling, and selective request processing methods. The methods mentioned above facilitate rapid response to traffic changes and ensure continuous service availability.

Scaling as a response to increasing load

Under conditions of variable network traffic, the scaling of computational resources is one of the fundamental approaches to ensuring fault tolerance and maintaining the performance of distributed systems. It enables infrastructure to adapt to current traffic volumes, thereby reducing the risk of service degradation when nominal throughput capacity is exceeded. In engineering practice, two principal types of scaling are distinguished: vertical and horizontal (table 1).

Table 1

Comparison of horizontal and vertical scaling [2, 3]

Parameter	Horizontal scaling (scale-out)	Vertical scaling (scale-up)
Principle	Adding new instances (nodes, pods, containers) that work in parallel and share traffic.	Increasing resources (CPU, RAM, I/O) of an existing node to handle more workload.
Use cases	Ideal for stateless microservices and independently scalable cloud-native components.	Applicable to monolithic applications and databases where request splitting is difficult.
Scalability limit	Not strictly limited; constrained by routing complexity and inter-instance coordination.	Limited by physical hardware boundaries (max CPU cores, memory capacity, I/O limits).
Availability impact	Typically zero downtime; new replicas spin up in parallel.	May require system reboot or redeployment, which can lead to temporary service downtime.
Fault tolerance	High resilience – failure of a single instance doesn't affect the system if proper load balancing is in place.	Lower resilience – node failure can halt service unless a high-availability cluster is configured.
Infrastructure needs	Requires orchestrators, load balancers (L4/L7), and service discovery tools for traffic routing and coordination.	Fewer external components needed, but demands high-performance hardware and may require manual tuning.

Although horizontal and vertical scaling both address variance in load and load variance through reactive resource provisioning, both approaches must base themselves on system state at the time and threshold-based metrics. However, where traffic variability spikes occur suddenly and unpredictably, reactive autoscaling may not be sufficient for guaranteeing service continuity. Distributed systems overcome this limitation by relying on smart autoscaling based on machine learning algorithms to anticipate traffic behavior and pre-empt accordingly. By predicting such traffic behavior ahead of time, infrastructure can pre-prepare for demand bursts and minimize latency, avoid resource overload, and maintain the risk of service degradation at low levels.

Rate limiting mechanisms

Controlling the rate of incoming requests is essential for maintaining the stability of distributed systems under high or unpredictable load. Several classical rate limiting algorithms exist, each exhibiting distinct characteristics in terms of resilience, control accuracy, and tolerance for short-term traffic bursts (table 2).

Table 2

Comparison of rate limiting algorithms in distributed systems [4, 5]

Algorithm	Mechanism	Behavior under load	Flexibility
Token Bucket	Tokens are generated at a fixed rate; each token permits one request. If tokens are available, requests are processed immediately.	Allows short-term bursts while maintaining a defined average request rate.	Moderate – configurable burst capacity and refill rate.
Leaky Bucket	Requests enter a fixed-size queue and are processed at a constant rate, regardless of input speed.	Strictly enforces rate limit; drops excess requests during peak load.	Low – cannot accommodate bursts.
Sliding Window	Tracks the number of requests within a moving time window (e.g., last 60 seconds).	Provides fine-grained rate enforcement and eliminates artifacts from fixed intervals.	High – dynamic window movement reflects actual load trends.

Algorithm	Mechanism	Behavior under load	Flexibility
Adaptive rate limit	Dynamically adjusts rate limits based on real-time signals such as latency, CPU load, and error rate.	Reacts to system stress – reduces throughput during overload, restores limits when system stabilizes.	Very high – can auto-tune thresholds per traffic behavior.

The choice of a rate limiting algorithm should be dictated by system design, traffic profiles, and fault tolerance requirements. In situations of steady load and stringent rate enforcement requirement, algorithms like Leaky Bucket or Sliding Window are more appropriate. Token Bucket offers greater flexibility to allow short-term bursts in traffic. Adaptive models are necessary in highly dynamic situations where user behavior is less likely to be predictable. Their integration with telemetry and observability systems is the foundation for intelligent traffic management policies that prevent performance loss and optimize resilience of distributed systems when exposed to loads.

Adaptive load shedding strategies for resource-constrained distributed systems

Handling excess demand under resource saturation requires deliberate mechanisms for selectively dropping traffic. Load shedding is one such controlled strategy, where a system intentionally discards part of the incoming requests when predefined performance thresholds are reached or exceeded. Unlike rate limiting, which regulates request flow during normal operation, load shedding is activated in response to actual or imminent overload.

One common method is **priority-based request filtering**, which classifies incoming traffic by criticality. Requests are given high, medium, or low priority according to business rules or service-level agreements (SLA). For example, order processing may be essential in e-commerce, while analytics or background synchronization can be backgrounded. Under overload, the system drops or delays lower-priority requests in order to preserve resources for critical operations. This requires priority routing and classification features at the API gateway or service mesh.

A second normal method is **queue-based admission control**, in which the new requests are placed in a queue and executed based on the available resources. When the queue crosses a certain threshold, the system can reject new requests or return backpressure to slow incoming traffic. This method smoothes out short-term bursts and distributes processing over time but may also require additional logic in the event of extended overload to prevent queue overruns and client timeouts.

Circuit breakers are crucial in service-to-service architecture by isolating directly failed or overloaded components to prevent cascaded failure. The circuit breaker trips to an «open» state when it reaches a threshold of errors, temporarily blocking calls to the failed service. Within this period, fallback responses may be provided, or traffic sent to backups. Following the cooldown time, the circuit enters a «half-open» condition to recover. In the case of a positive response, the connection is reopened. This fault-tolerance mechanism allows failing systems to remain operational without total failure.

The combination of these techniques – priority filtering, admission control, and circuit breakers – provides a multi-layered load shedding technique. Distributed systems can adaptively handle overload through this technique, maintaining stability and service continuity despite severe load.

Architectural realization of dynamic traffic management in cloud-native environments

Modern distributed systems are increasingly built using cloud-native approaches, where pliability, scalability, and manageability are achieved through dynamic resource management triggered by events and metrics. Cloud environments offer built-in mechanisms for implementing load balancing, rate limiting, load shedding, and autoscaling mechanisms. Their architectural model is based on microservices, containerization, orchestration, and service mesh technologies, creating a favorable environment for deploying traffic management policies with a high degree of automation and observability (table 3).

Table 3

Traffic management mechanisms in cloud-native architectures [6, 7]

Component / tool	Functionality	Level of control	Traffic management role	Example use case
Kubernetes HPA	Automatically adjusts the number of pods based on metrics (e.g., CPU, custom).	Application layer (Pod level)	Horizontal autoscaling – reacts to load changes to maintain system responsiveness.	Scaling stateless services under fluctuating web traffic.
Kubernetes VPA	Dynamically tunes resource requests/limits (CPU, memory) for running pods.	Resource scheduling layer	Vertical autoscaling – optimizes resource allocation per pod.	Adjusting memory for JVM-based services with variable heap usage.
Istio / Envoy Rate Limit	Enforces rate limiting policies using local or distributed quotas.	L7 (Service mesh / proxy level)	Throttles incoming traffic to prevent overload and protect downstream services.	Limiting login attempts or API call rates per user.
Envoy Circuit Breaker	Temporarily halts requests to failing services based on error thresholds.	L7 (within service mesh)	Prevents cascading failures by isolating unhealthy service instances.	Redirecting traffic during database unavailability.
Kubernetes Ingress + LB	Routes external traffic to services; supports path-based and host-based routing.	Edge (L4/L7, depending on setup)	Load balancing – distributes incoming traffic across service instances.	Balancing requests across replicated web services.

Such integration of mechanisms in cloud-native systems ensures not only component-level flexibility but also platform-level robustness across the system. Automated scaling, policy-driven routing with centralized control, and enterprise-scale telemetry support provide high controllability with low operational overheads. As a result, cloud platforms are not merely hosting infrastructure for services, but engaged agents for ensuring fault tolerance and flexibility in distributed digital infrastructure.

Observability-driven traffic control and failure mitigation in distributed systems

With highly volatile traffic and increasingly sophisticated microservice architecture, reactive methods such as autoscaling or load shedding alone are not sufficient to ensure the reliability of digital infrastructure in such environments. Credible and reliable functioning of distributed systems is only feasible with end-to-end observability across all layers – ranging from low-level networking interactions to high-level business metrics. With this paradigm, traffic control by observability is the inevitable next step in adaptive system development whereby not only is there response to what has already happened but also anticipation of potential overloads and the anticipation of failure prior to its occurrence.

Modern distributed systems generate huge volumes of telemetry data that typically happen in three flavors: metrics, tracing, and logging. Summarized numerical measures such as mean CPU usage, request rate, latency, and error rate are metrics [8]. They are real-time feedback signals employed for autoscaling, rate limiting, or load redistribution. Distributed tracing is required for detailed analysis, providing end-to-end visibility into request flows between microservices, indicating delays and bottlenecks at each step of processing. Logging provides the most detailed event-level data – capturing exceptions, warnings, and custom messages – and underlies anomaly detection and post-incident diagnostics.

Effective traffic management requires not only the collection of this telemetry, but its structured aggregation, correlation, and real-time analysis. Tools such as **Prometheus (metrics), Jaeger or**

OpenTelemetry (tracing), and Grafana Loki or ELK Stack (logs) enable a holistic view of infrastructure state and behavior. In more advanced configurations, observability data feeds machine learning pipelines for traffic prediction and anomaly detection.

By employing time-series data, traffic loads may be forecast using regression models, ARIMA, or neural networks such as LSTM. These prediction techniques may detect early warnings for upcoming overloads – for instance, high latency at sustained request rates or rising message queue lengths. Services may pre-emptively scale out, adjust rate limits, initiate circuit breakers, or redistribute traffic flows accordingly. This preventive measure has the effect of reducing occurrence rate of incidents and recovery time, and improving overall service quality.

Thus, observability-enabled traffic management shifts failure management from a reactive to proactive strategy. Through the incorporation of telemetry and predictive analytics in the architectural foundation, systems are endowed with self-healing and self-regulating characteristics, making them resilient against noisy workloads, software bugs, and external interference. It is essential in crafting mature, cloud-native digital infrastructures that can provide sustainable operational excellence.

Conclusion

With today's high-loaded and constantly evolving digital environment, fault tolerance and adaptability of distributed systems are impossible without the integrated employment of traffic management technologies. Load balancing, adaptive rate limiting, load shedding, horizontal and vertical scaling approaches enable systems to dynamically respond to fluctuating traffic volumes and prevent service degradation. Their integration with cloud platforms through the employment of tools like Kubernetes, Istio, Envoy, and KEDA ensures infrastructure level scaling and control over the operation. At the same time, observability acts as the catalyst, transforming the traffic control from reactive to proactive, self-adjusting behavior. All of these collectively ensure a fault-tolerant architectural environment for continuous operation in uncertainty and with stability, predictability, and quality of service persistence.

References

1. Ren Y., Wei M., Xin H., Yang T., Qi Y. Distributed network traffic scheduling via trust-constrained policy learning mechanisms // Transactions on Computational and Scientific Methods. 2025. Vol. 5. № 4.
2. Knyazeva A. Decentralized authentication methods for distributed networks // Professional Bulletin: Information Technology and Security. 2024. № 3/2024. P. 12-15.
3. Tsyganok R. Methodology for Building Scalable Microservice Architectures on Go for High-Load E-Commerce Platforms // Universal Library of Engineering Technology. 2025. Vol 1. № 2.
4. Peri A., Tsenos M., Kalogeraki V. Vertical Scaling Can Save Time: Optimizing Container Scheduling to Handle Sudden Bursts // Proceedings of the 19th ACM International Conference on Distributed and Event-based Systems. 2025. P. 86-97.
5. Kalyanasundaram T., Panchalingam K., Jegatheesan T., Wijayasiri A., Perera S. Load Balancer Filter-Based Approach To Enable Distributed API Rate Limiting // 2025 37th Conference of Open Innovations Association (FRUCT). IEEE. 2025. P. 75-85.
6. Blazhkovskii A. Collecting metrics for continuous platform monitoring // Universum: technical sciences: electronic scientific journal. 2025. № 3(132). P. 10-15.
7. Oyeniran O.C., Adewusi A.O., Adeleke A.G., Akwawa L.A., Azubuko C.F. Microservices architecture in cloud-native applications: Design patterns and scalability // International Journal of Advanced Research and Interdisciplinary Scientific Endeavours. 2024. Vol. 1. № 2. P. 92-106.
8. Zibitsker B., Lupersolsky A. Cost Optimization and Performance Control in the Hybrid Multi-cloud Environment // Proceedings of the 16th ACM/SPEC International Conference on Performance Engineering. 2025. P. 147-157.

UDC 004.438:004.42

INTEGRATION OF DEVOPS PRACTICES INTO DEVELOPMENT AND OPERATIONS PROCESSES OF RUBY APPLICATIONS

Topalidi A.
specialist degree, Moscow state university of geodesy and cartography (Moscow, Russia)

ИНТЕГРАЦИЯ DEVOPS-ПРАКТИК В ПРОЦЕССЫ РАЗРАБОТКИ И ЭКСПЛУАТАЦИИ ПРИЛОЖЕНИЙ НА RUBY

Топалиди А.В.
специалист, Московский государственный университет геодезии и картографии (Москва, Россия)

Abstract

This article examines the integration of DevOps practices into the development and operations processes of applications built with the Ruby programming language. Particular attention is given to the architectural and organizational characteristics of Ruby projects that influence the implementation of automated delivery, testing, and maintenance workflows. It explores the use of modern DevOps tools such as Docker, Kubernetes, and Terraform within the Ruby ecosystem, focusing on their role in ensuring environment reproducibility, service orchestration, and infrastructure as code management. The impact of these solutions on improving the resilience, scalability, and reliability of the application lifecycle is investigated, along with practical automation cases and typical implementation schemes.

Keywords: DevOps, Ruby, automation, Continuous Integration/ Continuous Deployment, Infrastructure as Code, Docker, Kubernetes, Terraform, application operations.

Аннотация

В данной статье рассматривается интеграция DevOps-практик в процессы разработки и эксплуатации приложений, созданных с использованием языка программирования Ruby. Особое внимание уделяется архитектурным и организационным особенностям Ruby-проектов, влияющим на внедрение автоматизированных процессов доставки, тестирования и сопровождения программного обеспечения. Изучается применение современных DevOps-инструментов, таких как Docker, Kubernetes и Terraform, в контексте Ruby-экосистемы, их роль в обеспечении воспроизводимости среды, оркестрации сервисов и управлении инфраструктурой как кодом. Исследуется влияние этих решений на повышение отказоустойчивости, масштабируемости и надежности жизненного цикла приложений, а также рассматриваются практические кейсы автоматизации и типовые схемы технической реализации.

Ключевые слова: DevOps, Ruby, автоматизация, Continuous Integration/ Continuous Deployment, инфраструктура как код, Docker, Kubernetes, Terraform, прикладные операции.

Introduction

Over the past years, significant changes have taken place in the methods used for software development and maintenance. With the demands for faster release cycles and greater scalability running higher, the adoption of DevOps practices has become of prime importance. Such integration allows for the automation of the life cycle of the software, the promotion of deployment stability, and the simplification of infrastructure management. Especially for applications created with Ruby, and

more so for the Ruby on Rails environment, the complementary use of DevOps tools becomes especially relevant with the specific characteristics of the ecosystem and with the dominance of monolithic architectures in place.

Today's DevOps software, such as Kubernetes, Terraform or Docker, have embedded features to standardize the environment and create Continuous Integration/Continuous Deployment (CI/CD) pipelines. In spite of that, the use of such software in Ruby-based projects requires proper attention to aspects such as managing dependencies, structuring code, and running the applications on various types of environments. Analyzing both successful implementations and common challenges associated with these technologies can inform more effective strategies for automation and infrastructure governance within the Ruby ecosystem. The goal of this research is to examine the specific aspects of integrating DevOps practices and tools, such as Docker, Kubernetes, Terraform into the development and operations processes of Ruby applications.

Main part. Peculiarities of integrating DevOps practices into the development and operation of Ruby applications

The adoption of DevOps practices in Ruby-based projects requires careful analysis of multiple architectural, organizational, and technological facets. Although Ruby has been one of the leading programming languages used for the development of web applications, its ecosystem has centred mostly around fast prototyping, monolithic structure, and in-situ configuration options. Such characteristics pose unique obstacles to the adoption of advanced DevOps practices like continuous integration, continuous delivery, infrastructure as code, and dynamic scaling.

Most Ruby applications are developed using the Model–View–Controller paradigm and follow a monolithic structure [1]. While this enables development and testing activities locally, it restricts flexibility in terms of automation of deployment tasks and the management of application components independently. The adoption of DevOps practices helps tackle these constraints through creating standardized pipelines and ensuring consistency of environments between development, testing, and production phases.

One of the first steps in adopting DevOps integration for Ruby projects is the use of CI tools. The Ruby language has strong support for testing and static code analysis, with frameworks like RSpec and Minitest being widely used for unit and integration testing. RuboCop and Brakeman are also being used for security auditing. Automation of checks at the CI phase ensures that defects are caught early, before changes are merged into the main codebase, thus reducing the chances of defects and improving the overall stability of the codebase. Typical CI pipelines in Ruby projects include dependency installation via Bundler, execution of tests and linters, security checks, and artifact generation, particularly the creation of Docker images in containerized environments [2].

A core principle of DevOps integration is the creation of a consistent runtime environment. Unlike with many programming languages, Ruby applications are more sensitive to interpreter version changes, dependency setups, and OS configurations. Therefore, environmental consistency must be ensured by the use of configuration files (like .ruby-version, .ruby-gemset, and Gemfile.lock) and automated provisioning tools (like Vagrant or containerized environments).

From an operational perspective, one of the persistent challenges in Ruby applications lies in managing application state and dependencies, particularly in scenarios involving background jobs and message queues. Many Ruby-based systems rely on asynchronous frameworks and libraries, such as Sidekiq and Resque, that operate in separate processes, requiring careful coordination between services and consistent lifecycle management. DevOps practices facilitate the automation of these tasks through deployment scripts, state monitoring, and orchestration of background workers [3].

Equally important is the cultural shift in collaboration between developers and operations engineers. The Ruby community has traditionally put more weight on code quality and design modularity; infrastructural concerns have always been considered somewhat secondary to these. The advent of DevOps patterns requires a redefinition of these boundaries, where the developer takes more responsibility for the pipeline configuration, the infrastructure definition in code, and the deployment into the environment.

The application of DevOps practices to Ruby applications is typically an evolutionary move away from old, locally focused development patterns towards a more mature and flexible system. The adoption of CI/CD pipelines, environment standardization, and the automation of testing and deployment processes has become an essential component of modern Ruby projects.

The application of Docker, Kubernetes and Terraform to Ruby projects: automation, infrastructure and continuous delivery

Modern application development practices force teams not just to code but also to handle the whole application lifecycle, including the build and test stages, along with the deploy ability to scale in the prod environments. On Ruby projects, particularly the ones under development and scaling, Docker, Kubernetes, and Terraform have become the essential tools to fulfill the demands for the above tasks. Their adoption enables environment reproducibility, centralized infrastructure management, and reliable implementation of CD pipelines.

Docker addresses one of the central challenges of Ruby applications – environmental dependency. Given Ruby's sensitivity to interpreter versions, library compatibility, and build toolchains, containerization offers a standardized configuration for application execution. Each component, including the Ruby runtime, required gems, build systems, and supporting utilities, is encapsulated within a Docker image. This methodology secures consistent behavior across development, testing, and production environments. Docker images can be transferred to registries and deployed across different environments without modification, significantly reducing errors caused by configuration inconsistencies.

Kubernetes is employed to manage scalability, fault tolerance, and complex environment configurations. Its use in Ruby-based projects enables flexible orchestration, load distribution, and automatic recovery in the event of failures. Monolithic applications are readily deployable in a Kubernetes cluster, where the different components like the Rails server, background job processor, database, caching store, and dependencies are envisioned and managed as separate entities. Kubernetes allows for careful management of resource allocation and provides automated scaling based on changes in workload intensity.

One of Kubernetes' most major strengths is the capacity to supply with uniform deployment to multiple environments such as testing, quality assurance, and production environments. The use of configuration files and templating software like Helm charts allows the recreation of infrastructure in a rigid manner on a diverse range of environments. Kubernetes further helps in the orchestration and maintenance of background tasks and auxiliary services like Sidekiq queues, WebSocket servers, and scheduled tasks that have traditionally presented some operating difficulties in Ruby-based infrastructures.

Docker and Kubernetes solve problems with the service orchestration and runtime environment. Nevertheless, Terraform has functionality to manage the underlying infrastructure. It is not rare to find the use of Terraform in Ruby projects to create the important components such as servers, databases, storage devices, load balancers, network devices, and others that form the base components. This process follows the infrastructure-as-code principles and guarantee reproducibility and ease of tracking changes.

Terraform is especially useful in situations where fast scalability or the creation of fresh environments are needed. Instead of handling cloud resources manually, groups are capable of creating reusable templates that only take slight adjustments for provisioning. This process greatly reduces the prospect of human error while speeding up the deployment of fresh features. Moreover, automated infrastructure management enables straightforward recovery from failures, as the entire architecture, codified and versioned, can be restored to its original state when necessary. To illustrate how these tools operate in conjunction throughout the application lifecycle, the figure 1 outlines a typical DevOps pipeline for a Ruby-based project.

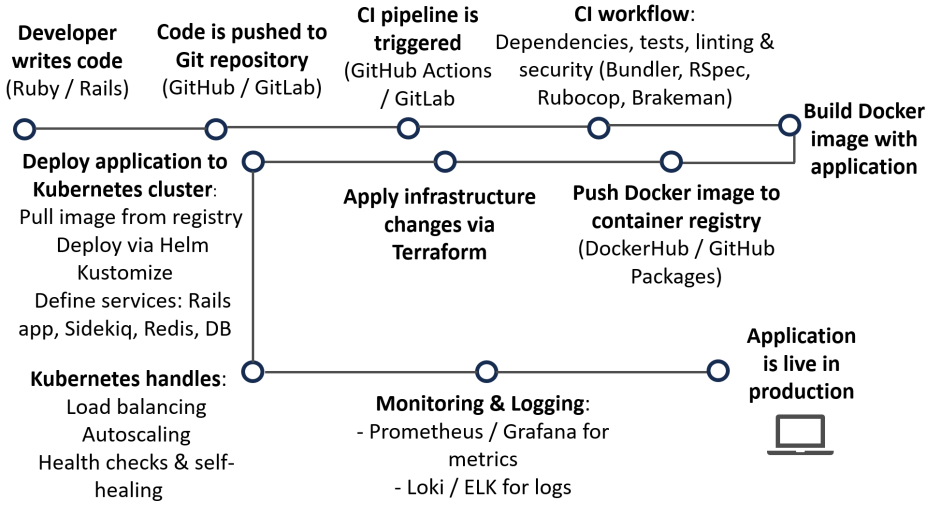


Figure 1. DevOps workflow for Ruby-based applications

Given the distinct roles and technical capabilities of listed tools within the DevOps ecosystem, it is important to consider their respective strengths and limitations when applied to Ruby-based projects (table 1).

Table 1

Comparison of DevOps tools [4, 5]

Tool	Advantages	Disadvantages
Docker	Environment isolation, simplified local development, consistent deployment across environments.	Network and permission complexities, requires optimized Dockerfile configuration.
Kubernetes	Automatic scaling and self-healing, high availability, centralized orchestration.	Steep learning curve, complex initial setup.
Terraform	Infrastructure as code, multi-cloud support, reproducibility and version control.	Risk of destructive changes, depends on provider stability and API.

The integration of Docker, Kubernetes, and Terraform into a unified DevOps pipeline enables Ruby projects to transcend traditional deployment models and embrace full automation and operational control. Such a pipeline improves process predictability, reduces manual intervention, accelerates release cycles, and facilitates frequent code changes without compromising system stability. These benefits are particularly evident in Agile-oriented teams, where rapid value delivery and responsiveness to changing requirements are critical [6].

Shopify is one of the world's largest e-commerce platforms that makes extensive use of Ruby on Rails and has successfully implemented a large-scale DevOps infrastructure based on Kubernetes. The engineering team developed an internal platform for orchestrating microservices and development environments, enabling the creation of automated deployment workflows and management of over two thousand services. Through deep integration of CI/CD pipelines and a well-defined infrastructure-as-code strategy, Shopify has achieved both high delivery velocity and stable environment provisioning while maintaining the core of its business logic in Ruby [7].

GitLab is a widely used DevOps lifecycle management platform and primarily written in Ruby and serves as a prominent example of deep DevOps integration within a Ruby-based project [8]. The internal architecture of GitLab supports containerization through Docker, automated CI/CD pipelines, and a scalable deployment layer on Kubernetes. In addition, GitLab leverages Terraform for cloud resource provisioning, allowing the team to deploy both testing and production environments programmatically.

Discourse is a popular open-source platform for online forums and also built with Ruby on Rails and was designed from the outset with deployment automation and operational efficiency in mind. The development team maintains an official Docker-based stack that enables one-command

deployment across a wide range of server infrastructures [9]. Automation extends to environment configuration, dependency updates, backups, and scalability operations.

Consequently, the adoption of modern DevOps tools allows Ruby applications to overcome limitations associated with legacy development and deployment practices, positioning them to meet the demands of contemporary high-load environments. Effectively implemented tools boost technological resilience and promote organizational process maturity, laying the foundation for sustained growth and long-term project changes.

Conclusion

The adoption of DevOps practices in Ruby application development and management marks the significant milestone towards achieving stable and scale-incline software delivery. Traditionally, the Ruby world has favored localized, monolithic architectures, but rising expectations for reliability and the need for rapid deployment call for the adoption of automation tools and efficient infrastructure management. The adoption of CI/CD principles, along with the unified approach for runtime environments, allows Ruby projects to get aligned with the latest software engineering practices.

The combination of Docker, Kubernetes, and Terraform increases organizational maturity and technical flexibility. Containerization, with collaborative working, promotes better reproducibility. Orchestration provides exact control and scale, and infrastructure as code makes the process more dependable and visible for change management steps. Together, all these elements constitute one complete technological stack where the development and operational procedures are highly coupled and bonded with the power of automation. While considering upcoming innovations, this integration has been considered as the vital base for efficient Ruby application development, examining the mounting complexity and higher performance expectations.

References

1. Milić M., Makajić-Nikolić D. Development of a quality-based model for software architecture optimization: a case study of monolith and microservice architectures // Symmetry. 2022. Vol. 14 (9). № 1824.

2. Dudak A., Israfilov A. Application of blockchain in IT infrastructure management: new opportunities for security assurance // German International Journal of Modern Science. 2024. № 92. P. 103-107.

3. Chatterjee P.S., Mittal H.K. Enhancing Operational Efficiency through the Integration of CI/CD and DevOps in Software Deployment // In2024 Sixth International Conference on Computational Intelligence and Communication Technologies (CCICT). 2024. P. 173-182.

4. Ponomarev E. Optimizing android application performance: modern methods and practices // Sciences of Europe. 2024. № 149. P. 62-64.

5. Chen C.C., Hung M.H., Lai K.C., Lin Y.C. Docker and Kubernetes // Industry 4.1: Intelligent Manufacturing with Zero Defects. 2021. P. 169-213.

6. Blazhkovskii A. Formation of high-performance teams in mobile development // Cold Science. 2025. № 13. P. 7-17.

7. Shopify-Made Patterns in Our Rails Apps / Shopify.Engineering // URL: https://shopify.engineering/shopify-made-patterns-in-our-rails-apps (date of application: 17.08.2025).

8. Infrastructure as Code with OpenTofu and GitLab / GitLab // URL: https://docs.gitlab.com/user/infrastructure/iac/ (date of application: 18.08.2025).

9. Discourse_docker / GitHub // URL: https://github.com/discourse/discourse_docker (date of application: 20.08.2025).

UDC 004.4

ARCHITECTURAL DESIGN PATTERNS FOR HIGH-LOAD SYSTEMS: PRINCIPLES, TOOLS, AND SCALABILITY CONSTRAINTS

Berezhnoy A.
bachelor's degree, Peter the Great St. Petersburg polytechnic university (St. Petersburg, Russia)

АРХИТЕКТУРНЫЕ ПАТТЕРНЫ ПРОЕКТИРОВАНИЯ ВЫСОКОНАГРУЖЕННЫХ СИСТЕМ: ПРИНЦИПЫ, ИНСТРУМЕНТЫ И ОГРАНИЧЕНИЯ МАСШТАБИРОВАНИЯ

Бережной А.А.
бакалавр, Санкт-Петербургский политехнический университет Петра Великого (Санкт-Петербург, Россия)

Abstract

The article takes into account architectural patterns used when building high-load distributed systems, such as CQRS, Event Sourcing, and Circuit Breaker. It analyzes their rules of operation, typical use cases, and shortcomings in terms of scalability, consistency, and fault tolerance. The paper presents a functional classification of these patterns, highlighting their engineering advantages and operational risks. Real-world implementation examples from major technology companies are provided, and criteria for selecting architectural solutions are formulated based on workload characteristics and infrastructure maturity. The article concludes by emphasizing the rationale for combining patterns within a unified architectural strategy, taking into account the specifics of business logic and system availability requirements.

Keywords: architectural patterns, high-load systems, CQRS, Event Sourcing, Circuit Breaker, scalability, fault tolerance.

Аннотация

В статье рассматриваются архитектурные паттерны, применяемые при проектировании высоконагруженных распределенных систем, такие как CQRS, Event Sourcing и Circuit Breaker. Анализируются принципы их работы, типовые сценарии использования и ограничения, связанные с масштабированием, согласованностью данных и отказоустойчивостью. Описаны функциональные классификации паттернов, их инженерные преимущества и эксплуатационные риски. Приводятся примеры применения в крупных компаниях и формулируются критерии выбора архитектурных решений в зависимости от характера нагрузки и зрелости инфраструктуры. Делается вывод о целесообразности комбинирования паттернов в рамках единой архитектурной стратегии с учетом специфики бизнес-логики и требований к доступности.

Ключевые слова: архитектурные паттерны, высоконагруженные системы, CQRS, Event Sourcing, Circuit Breaker, масштабирование, отказоустойчивость.

Introduction

Modern high-load information systems form an integral part of digital industry infrastructure in e-commerce, telecommunications, fintech, and logistics. These systems process millions of requests, ensure continuous availability of the service, and should prove to be resilient to all types of failure. In situations with sharply growing user activity and increasing complexity of business logic,

the architecture of a system is the decisive aspect in reliability and scalability. As such, most emphasis is on architectural design patterns that help deal with complexity, enhance fault tolerance, and make distributed systems more flexible.

The goal of this article is to provide an organized presentation of architectural patterns used when constructing high-load systems and their principles, supporting tools, and scalability limitations.

Main part. Fundamentals of high-load system design

High-load systems are distributed computing systems that are intended to process huge volumes of data and massive numbers of simultaneous requests with high reliability and minimal latency. They are applied in high-traffic setups with rigorous availability requirements, such as e-commerce, real-time money transactions, telecommunications, web streaming, and high-frequency data processing [1]. Their architecture must exhibit uniform performance at the peak point of load, flexibility of resources, and continuous scalability without degrading the quality of service.

The main technical characteristics of **high-load systems are horizontal scalability support, partial failure tolerance, low latency, and high throughput**. The current trending architectural practices involve stateless services, service decomposition using microservices, data replication, message queueing support (Apache Kafka), sophisticated load balancing techniques, distributed caching (Redis, Memcached), and data partitioning (sharding). Data consistency in distributed systems typically relies on eventual consistency models and CAP theorem trade-offs that call for transactional design and fault tolerance mechanisms.

Scalability is primarily achieved horizontally by adding new nodes and sharing traffic evenly across them. It includes close coupling of request routing, data locality, and minimizing inter-node communication. Fault tolerance is obtained through the duplication of essential elements, application of failure management patterns (Circuit Breakers, retries), real-time system monitoring, and automatic recovery assistance. High-load systems thus necessitate a unified solution that considers architectural principles, engineering methods, and automation tools.

Overview of architectural patterns

Architectural design patterns are reusable solutions to common problems that prove to be recurrent in software system development. Individual algorithms or libraries, by contrast, formalize successful architectural methods of structuring components, intercomponent communication, processing data, and failure recovery [2]. Though these templates do not provide for hard-coded implementations, they supply a strategic framework within which developers can bring system architecture into line with specific requirements.

There are numerous types of architectural patterns, but the most insightful within the high-load system setting is the functional orientation classification, i.e., the specific issues to be solved by the patterns (table 1).

Table 1

Functional classification of architectural patterns in high-load systems

Functional category	Purpose of patterns	Example use cases
State management	Efficient storage, recovery, and replication of data	Cache consistency, change history tracking
Fault isolation and resilience	Protection against cascading failures, process recovery	Timeout enforcement, automatic fallback mechanisms
Load balancing and request routing	Optimization of incoming traffic distribution	Load balancing across microservices, geo-distributed routing
Asynchronous processing and event queues	Offloading tasks from the main execution flow	Logging, delayed processing, bulk event publishing
Scalability	Horizontal scaling of system components	Service decomposition, data sharding
Responsibility segregation	Improved modularity, testability, and scalability	Separation of read/write concerns, modular system architecture

Functional category	Purpose of patterns	Example use cases
Consistency management	Coordination of consistent state in distributed systems	Eventual consistency, distributed transaction reconciliation

Thus, the architectural pattern functional categorization is a disciplined approach to creating a system based on the type of problems they are designed to solve and where they operate. It is highly applicable in high-load systems, where each decision concerning architecture will have to meet strong requirements for performance, resilience, and scalability. The above-mentioned table serves as a reference when selecting the appropriate patterns to assist in creating well-proportioned and flexible architectures.

CQRS pattern as a mechanism for scalable and isolated management of read and write operations in high-load systems

The CQRS pattern (Command Query Responsibility Segregation) is an architectural approach in which state-modifying operations (commands) and read operations (queries) are handled by separate components (fig. 1).

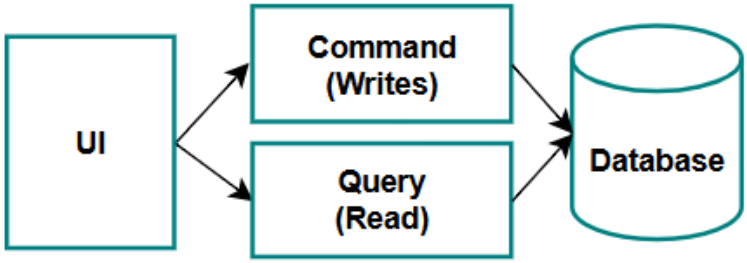


Figure 1. CQRS pattern

Such a design is based on separation of concerns: read and write operations require different data models, pose different performance and scaling requirements, and their combined implementation typically produces architectural baggage and violation of the Single Responsibility Principle (SRP).

The use of the CQRS pattern in high-load system design is appropriate where there are asymmetric loads against read and write operations, business logic complexity, and independent scalability needs for system components. Its implementation should, however, be done with proper consideration of its engineering benefit and inherent trade-offs (table 2).

Table 2

Technical advantages and limitations of the CQRS pattern [3, 4]

Advantages	Disadvantages	Applicability notes
Independent scaling of read and write models using separate nodes, databases, or services	Increased architectural complexity, more services and sync logic required	Suitable for systems with read-heavy loads (e.g., 80–90% reads)
Optimized query performance via precomputed projections or denormalized views	Eventual consistency issues between write and read models	Effective in analytical or dashboard-heavy applications
Isolated business logic in command handlers improves maintainability and rule enforcement	Harder to trace failures due to asynchronous event flows	Works well with strict domain modeling (DDD); requires advanced monitoring
Flexible read models allow multiple data views for APIs or UI, without affecting write logic	Migration from CRUD systems is complex and may require data model redesign	Best for greenfield systems or new microservices
Read availability under partial failure, since read model can remain operational if write path fails	Higher DevOps overhead: separate deployments, queues, and fault tolerance setups	Critical for 24/7 systems requiring high availability

Advantages	Disadvantages	Applicability notes
Natural fit with Event Sourcing: easy to replay events and rebuild projections	Steeper learning curve for teams unfamiliar with distributed and event-driven systems	Suitable for experienced teams in complex domains

Thus, the CQRS pattern is an effective method of improving scalability, maintainability, and fault tolerance of high-loaded systems, particularly where read operations significantly surpass write operations and business logic requires strict processing isolation. However, its application introduces architectural and operational complexity and, consequently, this method is relevant primarily when the technical team is mature enough and there is an explicit need for the segregation of read and write concerns.

Event Sourcing pattern: state management through event streams in distributed systems

Event Sourcing pattern leverages the concept of keeping all changes to a system's state as an immutable stream of events. The system doesn't keep the entity's current state within the database but keeps every event that has shaped that state, in the order in which they occurred. Thus, the state at any point in time can be recreated through a re-run of the sequence of events from beginning. Such an approach makes tracing historical development precise, while simultaneously enhancing flexibility in modifying business logic, auditing, and system recovery (fig. 2).

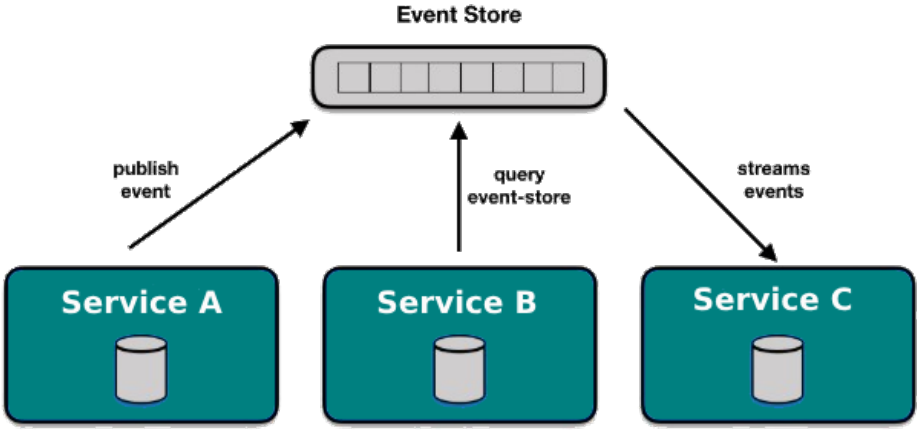


Figure 2. Event Sourcing pattern

It is particularly applicable in high-load systems where precise change tracking, operation rollbacks, full auditability, and process reproducibility are required. Event Sourcing is used in systems with complex business logic – such as financial systems, CRM systems, and logistics systems – as well as in architectures where data consistency between multiple services is needed. It is typically combined with CQRS: commands trigger event creation, which is then processed to build projections in read models. Table 3 lists the advantages and disadvantages of using Event Sourcing in high-load architectures.

Table 3

Advantages and limitations of the Event Sourcing pattern in high-load architectures [5, 6]

Advantages	Disadvantages	Applicability notes
Complete change history: every event is stored, ensuring full traceability and auditability.	Storage complexity: requires designing an event store and supporting event versioning.	Relevant for domains with strict regulatory or audit requirements (e.g., finance, insurance).
High fault tolerance: state can be restored by replaying past events.	Consistency and ordering issues in distributed systems.	Requires idempotency strategies and control of event order.
Ability to rollback or simulate changes (event replay, time-travel debugging).	Increased development complexity: event schema migrations and aggregate management are required.	Suitable for experienced teams working with event-driven architectures.

Advantages	Disadvantages	Applicability notes
Easy integration with CQRS: events directly feed projections in read models.	Large data volume due to the accumulation of events over time.	Requires snapshotting or archival strategies to manage storage.
Flexible business logic: a single event can be processed by multiple subscribers.	Demands robust event delivery and consistency infrastructure (brokers, retry logic).	Requires a reliable event backbone such as Kafka, NATS, or Pulsar.
Improved scalability and asynchronous processing.	Debugging and testing complexity: state is built dynamically, not centrally stored.	Requires visualization tools and event stream tracing support.

Hence, Event Sourcing is a good architectural style that provides great control over state management, complete change history, and flexible business logic evolution. It is particularly necessary in systems where strong audit requirements, reproducibility, and data consistency among microservices are required. However, adopting this pattern also requires specialized event-driven infrastructure, careful event modeling, and readiness to deal with the added complexity of debugging, scaling, and maintaining the system. The use of Event Sourcing should be deliberate and linked to actual system requirements rather than architectural trends.

Circuit Breaker pattern: failure management in distributed high-load systems

The Circuit Breaker pattern is employed to protect systems from cascaded failures and performance deterioration when auxiliary services are momentarily unavailable or loaded, respectively, for some elements. Its spirit is to monitor the status of the communications between services and programmatic call termination to flaky dependencies following a failure quota being reached. This prevents the failed component from being overloaded, reduces response time latency, and maintains the integrity of the remaining system (fig. 3).

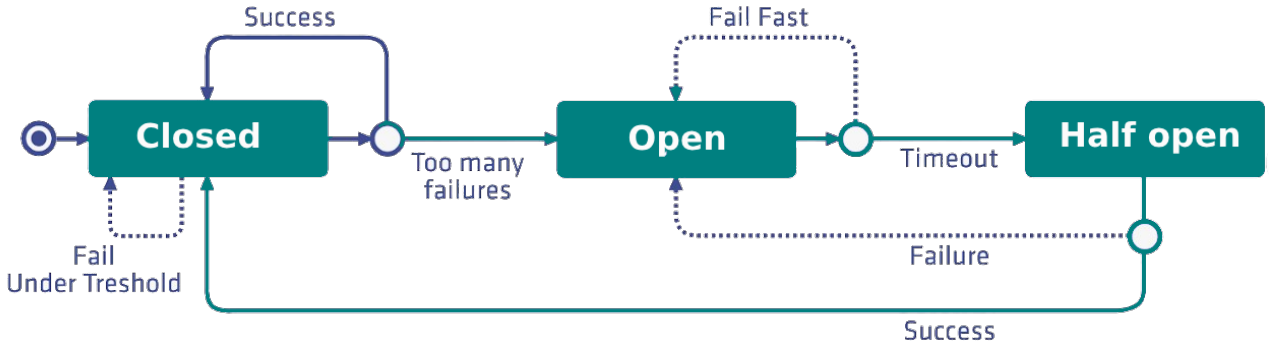


Figure 3. Circuit Breaker pattern

In high-load system architecture, the Circuit Breaker is a crucial component within fault-tolerance mechanisms, implemented on the basis of the fail-fast principle. It works best in microservice and distributed systems, where numerous services communicate across the network, and delays or failures in one component can lead to a domino effect. The pattern is implemented as a mid-layer between calling and called components, transitioning between the following states automatically: closed (normal operation), open (calls are disallowed), and half-open (availability is tested). As in the case of the patterns discussed above, Circuit Breaker has its own characteristics and trade-offs (table 4).

Table 4

Advantages and limitations of the Circuit Breaker pattern [7, 8]

Advantages	Disadvantages	Applicability notes
Reduces load on failing components by cutting off calls after reaching the failure threshold.	Requires careful configuration of error thresholds, timeouts, and recovery intervals.	Effective when interacting with unstable external APIs or services.

Advantages	Disadvantages	Applicability notes
Lowers client-side latency during failures through fail-fast responses.	Risk of premature interruption due to temporary fluctuations in service performance.	Needs metrics analysis and proper sensitivity tuning.
Increases overall system stability by isolating failures.	Requires continuous monitoring and logging of Circuit Breaker states.	Suitable for microservice architectures with many service dependencies.
Supports automatic recovery via the half-open state.	Complex to test under failure and recovery scenarios at scale.	Integration with observability stack (e.g., Prometheus, Grafana, Zipkin) is essential.
Improves user experience by enabling controlled error handling and fallbacks.	May require additional infrastructure or libraries (e.g., Hystrix, Resilience4j).	Can be embedded in API gateways, service meshes, or middleware layers.
Easy to integrate into call chains using middleware or interceptors.	Increases tracing complexity in distributed environments.	Requires centralized logging and correlation of request identifiers.

The Circuit Breaker pattern is thus a fundamental building block for the design of resilient distributed systems by isolating failing components and preventing the propagation of errors. Its effectiveness, however, is directly tied to both the correctness of its setup as well as the quality of system observability. Properly implemented, not only does it improve fault tolerance, but it also deals with user experience and system responsiveness during partial outages. However, to be successfully adopted, it needs to integrate with monitoring tools and a mature operational environment that can support automated recovery and graceful degradation.

Applicability analysis in real-world systems

Architectural patterns for use in designing high-load distributed systems truly find their worth only if appropriately tailored to a specific deployment context. Selecting an appropriate strategy needs to depend on several factors like the type and magnitude of load for a system, the level of acceptable data consistency, response time requirements, and scalability of individual components. These patterns are applicable across various domains – including finance, telecommunications, e-commerce, and logistics – where reliability and throughput are critical [9]. No less important are the maturity of the team, the presence of a stable DevOps foundation, and readiness to handle sophisticated event-driven logic under production. Early adoption of a pattern, even one that is theoretically correct, on unjustified premises, may introduce with it increased operational complexity, reduced system reliability, and debug and maintenance challenges.

In reality, CQRS (Command Query Responsibility Segregation), Event Sourcing, and Circuit Breaker patterns are used in the industry on a wide scale to implement robust and scalable distributed systems. For instance, CQRS is used by **Amazon** in a number of high-volume, consumer-facing systems such as the Amazon Shopping Cart and Order Management System, where read operations (e.g., availability of product, price, history of order) far exceed writes. By separating the read and write responsibility, Amazon is able to have distinct read-optimized and write-optimized data models – i.e., DynamoDB for write and ElasticSearch for query. This kind of architecture enables horizontal scaling of read replicas, reduces customer query latency during traffic surges (e.g., Black Friday), and enables event-driven propagation of updates to downstream services like billing or inventory. This design also enhances fault isolation and allows Amazon to modify read and write subsystems separately, reducing deployment risk and operational coupling.

Similarly, **Netflix** employs the Circuit Breaker pattern – especially via Resilience4j and the deprecated Hystrix – as an essential element of its microservice framework. For example, in the Netflix API Gateway layer, Circuit Breakers oversee outgoing requests to services downstream like the Recommendation Engine or User Profile Service. If latency surpasses set limits or error rates surge (for instance, from regional outages or backend issues), the breaker shifts to an «open» state, immediately failing calls to prevent overloading the target service. This fail-fast approach not only

shortens recovery time by decreasing load but also stops thread pools and connection limits in upstream systems from being exhausted.

Real-world experience with the subject matter proves that the best outcomes are achieved by carefully combining these patterns, supported by correct configuration and ongoing monitoring. CQRS, Event Sourcing, and Circuit Breaker complement each other by providing flexibility, reliability, and scalability. Their effective use, however, requires mature engineering culture, i.e., correct monitoring, distributed tracing, automated tests, and a correctly managed approach to evolving business logic over time.

Conclusion

Given the exponential growth of online services and the resulting increase in system load, the adoption of architectural patterns has become a fundamental requirement for ensuring system scalability, fault tolerance, and maintainability. CQRS, Event Sourcing, and the Circuit Breaker are patterns that perform best when properly applied in high-load system architecture, particularly in conjunction with event-based models and high-quality monitoring tools. Their use allows systems to evolve adaptively to new needs while reducing the effect of failures and uneven traffic. But effective use of these patterns does need mature engineering practices, well-thought-out architectural design, and detailed consideration of operational constraints.

References

1. Bolgov S. Development of high-load backend systems for banking products: problems and solutions // Proceedings of the LIII International Multidisciplinary Conference «Innovations and Tendencies of State-of-Art Science». Mijnbestseller Nederland, Rotterdam, Nederland. 2025. P. 44-51.

2. Perera C. Optimizing Performance in Parallel and Distributed Computing Systems for Large-Scale Applications // Journal of Advanced Computing Systems. 2024.Vol. 4. № 9. P. 35-44.

3. Cherif A.N., Youssfi M., En-naimani Z., Tadlaoui A., Soulami M., Bouattane O. CQRS and Blockchain with Zero-Knowledge Proofs for Secure Multi-Agent Decision-Making // International Journal of Advanced Computer Science & Applications. 2024. Vol. 15. № 11.

4. Youssfi M., Ezzrhari F.E., Hajoui Y., Bouattane O., Kaburlasos V. Multi-Micro-Agent System middleware model based on event sourcing and CQRS patterns // Smart Trajectories. CRC Press. 2022. P. 25-46.

5. Dhanaraj A. Building Resilient Systems: Error Handling, Retry Mechanisms, and Predictive Analytics in Event-Driven Architecture // Journal of Computer Science and Technology Studies. 2025. Vol. 7. № 7. P. 317-324.

6. Smirnov A. Efficient microservices scaling: Kubernetes, autoscaling, and load balancing // International Journal of Advances in Computer Science and Technology. 2025. Vol. 14(6). P. 22-24.

7. Punithavathy E., Priya N. Auto retry circuit breaker for enhanced performance in microservice applications // International Journal of Electrical & Computer Engineering (2088-8708). 2024. Vol. 14. № 2.

8. Iurchenko A. Optimization of Microservices Architecture Performance in High-Load Systems // The American Journal of Engineering and Technology. 2025. Vol. 7. № 05. P. 123-132.

9. Kovalenko A. Architectural and algorithmic methods for enhancing the resilience of high-load backend services in the financial sector // Norwegian Journal of development of the International Science. 2025. № 158. P. 87-91.

UDC 004.94: 336.64

DIGITAL VISUALIZATION OF INVESTMENT ACTIVITY IN THE EOS (VAULTA) BLOCKCHAIN ECOSYSTEM

Ulyanov V.
bachelor's degree, Azerbaijan state oil and industry university (Baku, Azerbaijan)

ЦИФРОВАЯ ВИЗУАЛИЗАЦИЯ ИНВЕСТИЦИОННОЙ АКТИВНОСТИ В БЛОКЧЕЙН-ЭКОСИСТЕМЕ EOS (VAULTA)

Ульянов В.В.
бакалавр, Азербайджанский государственный университет нефти и промышленности (Баку, Азербайджан)

Abstract

This article examines the digital visualization of investment activity in the EOS (Vaulta) blockchain ecosystem. The architecture of the platform based on the Delegated Proof of Stake mechanism is analyzed, as well as the specifics of the investment environment shaped by the resource model and the structure of incentives. Directions of financial activity and their impact on the behavior of ecosystem participants are studied. Special attention is paid to the structure of blockchain data and its analytical potential, as well as to the role of interface and visual-analytical tools in the interpretation of investment processes. User experience and interface are analyzed, ensuring cognitive accessibility of information, reduction of user workload, and increased trust in decentralized services.

Keywords: digital visualization, EOS (Vaulta), investment activity, blockchain data, interface tools, tokenomics.

Аннотация

В данной статье рассматривается цифровая визуализация инвестиционной активности в блокчейн-экосистеме EOS (Vaulta). Анализируется архитектура платформы, основанная на механизме Delegated Proof of Stake, а также специфика инвестиционной среды, формируемой под влиянием ресурсной модели и структуры стимулов. Исследуются направления финансовой активности, а также их влияние на поведение участников экосистемы. Особое внимание уделяется структуре блокчейн-данных и их аналитическому потенциалу, а также роли интерфейсных и визуально-аналитических инструментов в интерпретации инвестиционных процессов. Анализируются пользовательский опыт и интерфейс, обеспечивающие когнитивную доступность информации, снижение нагрузки на пользователя и повышение доверия к децентрализованным сервисам.

Ключевые слова: цифровая визуализация, EOS (Vaulta), инвестиционная активность, блокчейн-данные, интерфейсные инструменты, токеномика.

Introduction

The development of blockchain technologies has led to the formation of new forms of investment behavior, where transparency, accessibility, and data structuring play an important role. In the context of high complexity of network processes, one of the priority tasks is the creation of interface and analytical tools that allow not only tracking transactions and economic metrics but also interpreting them in the context of user behavior.

The EOS (renamed on May 14, 2025 to Vaulta in connection with the transition to a new strategy focused on creating Web3 banking) ecosystem is a representative model for analyzing such

processes due to the combination of the Delegated Proof of Stake (DPoS) mechanism and a developed application network. Visual and interface solutions acquire particular significance in it, providing both technical detailing of transactional flows and cognitive accessibility of analytics for different categories of participants. The purpose of the study is to analyze the role of interface and visual-analytical tools in the interpretation of investment activity in the EOS (Vaulta) ecosystem.

Main part. Investment environment of the EOS (Vaulta) ecosystem

The **EOS (Vaulta) ecosystem** occupies a special position among blockchain platforms focused on high-performance decentralized applications (dApps). It provides a scalable, almost instantaneously responsive, and economically efficient infrastructure. Its investment environment is shaped by architectural decisions, the governance mechanism, the structure of incentives, and the behavior of various participants.

At the core of EOS (Vaulta) lies the DPoS consensus mechanism. Unlike the traditional Proof of Work (PoW), where the right to create a block is obtained through computational costs, or the conventional Proof of Stake (PoS), which assumes a random selection of validators, it operates through delegated voting, in which token holders elect 21 block producers. These nodes act on behalf of the community, ensuring decentralization of governance while maintaining high throughput (fig. 1).

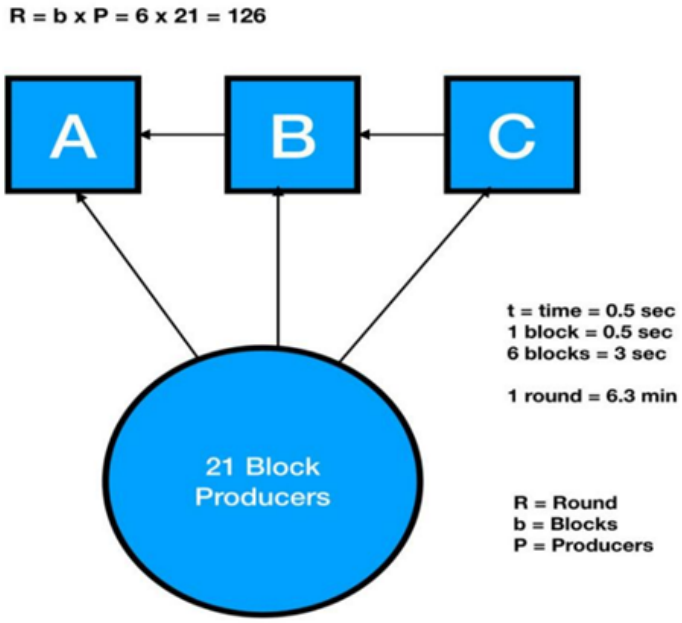


Figure 1. Architecture scheme of the DPoS mechanism in the EOS (Vaulta) network

The abandonment of traditional transaction fees fundamentally changes the nature of user experience (UX) and investment models. In this ecosystem, a resource model is applied, meaning that users reserve CPU, NET, and RAM required for executing transactions. This eliminates direct fees for operations but introduces variable costs associated with the fluctuating value of resources, creating a secondary market for computing power and storage. In practice, this directly affects investor strategies. Projects with high transaction frequency or significant data volumes are forced to take into account the dynamics of resource prices as a factor of operational risk and investment attractiveness.

The combination of delegated consensus, the resource model, and the absence of fees creates a unique economic environment with specific incentives. Under these conditions, the structure of interactions among ecosystem participants emerges (fig. 2).

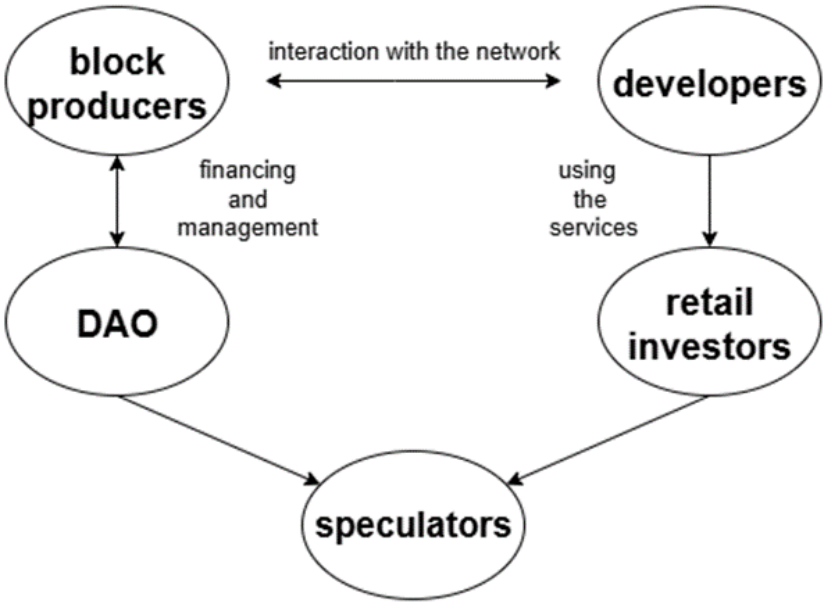


Figure 2. Role map in the EOS (Vaulta) ecosystem

Block producers, such as EOS Nation, EOS Authority, and Greymass, play a significant role by ensuring network stability and participating in its governance. Developers create dApps and maintain the infrastructure, shaping the innovative potential of the ecosystem. A decentralized autonomous organization (DAO) serves as a tool for collective governance and resource distribution. Retail investors focus on trading, staking, and participation in Initial DEX Offerings (IDO), while speculators respond to short-term market fluctuations, amplifying price dynamics and liquidity.

Investment flows in the ecosystem are directed not only to traditional segments such as token trading and staking but also to more complex structures. These areas differ in functionality, user engagement dynamics, and the maturity of financial instruments (table 1).

Table 1

Main directions of financial activity in the EOS (Vaulta) ecosystem [1]

Direction	**Functional features**	**Examples**	**The nature of investment activity**
Decentralized finance (DeFi)	Token exchange, staking, farming, stablecoin issuance, algorithmic lending.	Defibox, USN, sEOS	High, active trading, high liquidity.
Non-fungible token (NFT)	Creation and sale of digital objects (art, game items), collecting.	AtomicHub, EOSNFT, SimpleAssets	Average, undulating, depends on market trends.
DAO	Voting, fund management, decision-making by network users.	Eden on EOS (Vaulta), EOS DAC	Increasing, the involvement of token holders in management.
Tokenomics and staking	Staking tokens, voting for producers, receiving dividends, and participating in income distribution.	EOS (Vaulta)	Moderately high, long-term strategies and token retention.
Gaming and metaverse assets	Integration of in-game tokens, NFT objects, the internal economy of blockchain games.	Upland	Low-medium, depends on community activity.

Thus, the investment environment of the EOS (Vaulta) ecosystem represents the result of the interaction of architectural innovations, the delegated consensus mechanism, and the resource model with social governance practices and the diversity of economic strategies. Such a combination ensures high network throughput, stimulates the development of decentralized applications, and creates conditions for the participation of various categories of users.

Structure of blockchain data and their analytical potential

The study of investment activity in distributed ecosystems is impossible without understanding how the data forming the basis of the ledger are organized and processed. In the case of EOS (Vaulta), the volume of generated information is characterized by high density, continuous variability, and hierarchical organization. In addition to transaction records, the data include the results of smart contract execution, voting parameters, delegated powers, and the dynamics of account balance states (fig. 3).

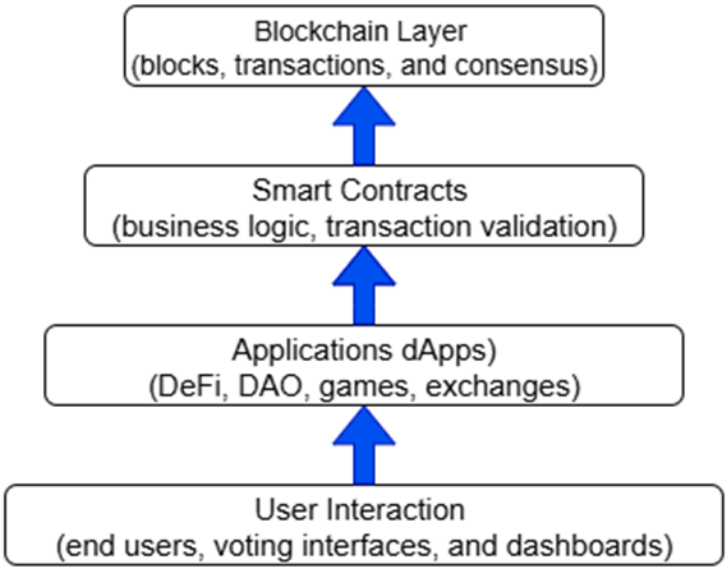


Figure 3. Multilayer data structure in EOS (Vaulta)

At the basic level**, blockchain data** represent transactions that record the transfer of digital assets between network participants. However, in EOS (Vaulta), transactions often serve only as an external interface for more complex operations, which conceal calls to smart contracts. Unlike bitcoin, where the data structure has limited functionality, it supports **multiple contracts calls within a single transaction**, creating specific conditions for analysis.

Alongside transactional activity, an important source of data is the **delegation and voting** system implemented within the DPoS mechanism. Every EOS (Vaulta) token holder has the ability to delegate their votes for the election of block producers, which is recorded in the public chain as specially formatted operations. These data make it possible to analyze the level of political activity in the ecosystem, the concentration of votes around specific validators, and to identify correlations between investment flows and changes in the governance configuration. For example, when leading block producers change, one can observe the outflow of tokens from their staking addresses and increased activity of competing nodes, which provides a basis for building behavioral models of investors.

However, the interpretation of these data is accompanied by serious difficulties. Information is often presented in a **fragmented** manner, since records are distributed among smart contracts and accounts without obvious links, which complicates their consolidation into a coherent picture. An additional problem is the **scale of the data**, as hundreds of thousands of transactions are recorded daily, and without automated filtering they lose their analytical significance. In this regard, **systematization and visual aggregation** of information become especially important, allowing for improved accuracy of interpretations and reduced time for extracting insights. Thus, the data structure of EOS (Vaulta) possesses high analytical potential, but it is fully revealed only under the condition of proper visual processing and contextual interpretation.

Interface and analytical solutions in the EOS (Vaulta) ecosystem

The development of the digital infrastructure of blockchain platforms goes beyond data transmission protocols and consensus mechanisms. A significant indicator of the ecosystem's maturity is the availability of interface and analytical tools that ensure accessibility and transparency of information (table 2).

Table 2

Interface and analytical tools [2]

Type of tool	Appointment	The effect on the ecosystem
Monitoring panels	Real-time tracking of key metrics.	Increasing transparency and manageability of processes.
Data visualization systems	Graphical representation of complex indicators.	Simplify the analysis and interpretation of information.
Interactive reports	Dynamic formation of analytics.	Adapting data to different user groups.
Navigation tools	Organization of access to information.	Ensuring the convenience and accessibility of interaction.

Thus, these tools represent the primary level of organizational support for the ecosystem. This approach is practically demonstrated by the **Everipedia (IQ Network)** project, where visual and interface solutions are used to organize decentralized access to knowledge, confirming the importance of convenient interfaces and analytics for engaging users in the ecosystem [3]. It is important to note that their significance is reinforced when combined with functional capabilities that specify ways of working with data and ensure the practical applicability of solutions (table 3).

Table 3

Functional capabilities of EOS (Vaulta) interface platforms [4, 5]

Function	Content	Application example
Filtering	Data selection by transaction type, contract address, time periods.	Analyzing the dynamics of operations with a specific smart contract.
Aggregation	Summarizing and grouping data to form a generalized picture.	Estimation of the total volume of votes or distribution of tokens.
Personalization	Configuring the display and saving of custom templates.	Subscribe to updates at selected addresses or integrate wallets.

These functions form the basis of analytical work with EOS (Vaulta) blockchain data. They make it possible to move from fragmented transaction records to a comprehensive picture of activity, simplifying the identification of patterns and providing the possibility of customizing analysis for specific research or investment tasks.

The logical continuation of this approach is the transition to more complex metrics, where the focus shifts from individual transactions to systemic indicators. In this context, the **visualization of EOS (Vaulta) ecosystem tokenomics** becomes particularly important. It covers such indicators as emission dynamics, the distribution of tokens among different categories of holders, and the level of inflation. The inclusion of these metrics in analytical dashboards makes it possible to align individual participants' strategies with system-wide processes and to assess the long-term sustainability of the network [6].

In practice, it performs the function of integrating heterogeneous data, allowing the transition from local observations to a macroeconomic level of analysis. Emission indicators reflect the pace of monetary supply expansion and provide the basis for assessing inflationary pressure. The distribution of tokens among retail investors, large holders, exchanges, and DAO indicates the degree of capital concentration and the level of decentralization. The share of tokens in staking and their turnover rate serve as indicators of trust in the ecosystem and the willingness of participants to lock funds for the long term.

Thus, tokenomics becomes a link between the technical infrastructure and the economic behavior of users. It makes it possible to identify signals of potential system overload, reduced investment attractiveness, or, conversely, network resilience under conditions of growing transactional activity.

The further development of the EOS (Vaulta) ecosystem involves adapting interfaces to the needs of different categories of users. For retail participants, the priority remains the simplicity and clarity of displaying balances, transaction histories, and staking parameters, while validators focus on access to voting statistics, block production efficiency, and network load. The organizers of DAO

require analytical tools to assess community activity, resource allocation, and the monitoring of voting processes. Taking these differences into account fosters broader involvement in governance mechanisms and increases the resilience of the entire system. Thus, interface and analytical tools form the foundation of EOS (Vaulta) investment and management environment, ensuring process transparency and consistency between the technical and economic levels of network functioning.

The role of UX/UI approaches in the visual analytics of the EOS (Vaulta) ecosystem

Modern blockchain analytics increasingly depends not only on the depth of processed data but also on the quality of its presentation to the end user. The effectiveness of UX and user interface (UI) solutions determines not only the convenience of interaction but also the level of trust in the platform, the degree of analytical engagement, and the user's willingness to make investment decisions [7]. Visual analytics here acts as an integral part of the overall trust architecture, and its effectiveness directly depends on how well the logic of perception, navigation, and interpretation is structured.

In the context of a multilayered blockchain data structure, each interface element must be organized in such a way that the user can easily **distinguish levels of information significance**. For example, when viewing block producer voting, producers with the highest number of votes are visually highlighted, while secondary data are displayed in the background without overloading perception.

Navigation, as a tool for moving through the interface, must be **predictable and intuitive**. In the EOS (Vaulta) ecosystem, navigation solutions are often implemented through multifunctional panels, tabs, and dynamic filters. An important condition for a modern interface is **adaptability**, meaning the system's ability to work equally well across different devices and screens.

However, the technical organization of the interface is only the foundation. Of equal importance is the cognitive component, namely the consideration of how humans perceive information. One of the main tasks in this context is the **reduction load**, especially when working with multidimensional and fragmented data. Visual patterns and metaphors play an important role here.

Interactivity is becoming increasingly important. Platforms provide users with the ability to choose the level of detail in displayed charts: from overall trading volume to the structure of a liquidity pool by tokens. Such customization of the interface makes visual analytics personalized, which is especially important in the context of constantly changing investment strategies. When users can determine for themselves which data are a priority and how they will be displayed, their engagement and the effectiveness of their interaction with the platform increase.

Finally, in the context of the blockchain environment, where the level of trust directly influences investor behavior, the **quality of UX becomes one of the factors in shaping loyalty and readiness for investment actions**. If the interface is difficult to use, overloaded, or unclear, this not only reduces the effectiveness of analysis but also creates a negative perception of the entire platform. Conversely, intuitive, visually clean, and responsive interfaces contribute not only to user retention but also to increasing the likelihood of their active participation in the economic and governance mechanisms of the ecosystem.

A telling example is the work of the American team **Greymass**, which developed the Anchor wallet, where intuitive interfaces were implemented for interacting with transactions, smart contracts, and voting for block producers [8]. This case demonstrates that UX/UI-oriented solutions enhance not only the convenience and security of working within the EOS (Vaulta) ecosystem but also its long-term resilience.

Thus, UX/UI approaches in the visual analytics of the EOS (Vaulta) blockchain play not an auxiliary, but a system-forming role. The success of the entire analytical environment, user trust, and the viability of the decentralized economy depend on how deeply they are integrated into the platform's architecture.

Conclusion

Interface and visual-analytical tools of the EOS (Vaulta) ecosystem play an important role in the interpretation of investment activity. Their significance is not only in enabling the transparency of process in transactions but also in the potential to arrange data on tokenomics, voting, and user behavior, thereby providing a full picture of the functioning of the decentralized network.

Visualization of economic and technical parameters enables the participants to make more informed decisions, which directly affects the sustainability of the ecosystem.

Solutions in the field of UX/UI acquire particular importance, as they determine the quality of interaction with analytical platforms. Intuitive navigation, flexibility, and responsiveness of interfaces reduce cognitive effort and build trust in systems, leading to increased user engagement. Interface solutions and visual analytics thus constitute a vital part of EOS (Vaulta) investment and management ecosystem, and their further development in the direction of personalization and openness will determine the long-term sustainability and competitiveness of decentralized platforms.

References

1. Shinkevich A.I., Kudryavtseva S.S., Samarina V.P. Ecosystems as an Innovative Tool for the Development of the Financial Sector in the Digital Economy // Journal of Risk and Financial Management. 2023. Vol. 16. № 2. P. 72.
2. Ehrensperger R., Sauerwein C., Breu R. A Maturity Model for Digital Business Ecosystems from an IT Perspective // Journal of Universal Computer Science (JUCS). 2023. Vol. 29. № 1. P. 34-72.
3. Maunu J. Revenue models of decentralized applications: an empirical study how decentralized software products generate income. 2025.
4. Liu H., Mao Y., Li X. An Empirical Analysis of EOS Blockchain: Architecture, Contract, and Security // ArXiv preprint arXiv: 2505.15051. 2025.
5. Kovalenko A. Architectural and algorithmic methods for enhancing the resilience of high-load backend services in the financial sector // Norwegian Journal of development of the International Science. 2025. № 158. P. 87-91.
6. Moncada R. Blockchain tokens, price volatility, and active user base: An empirical analysis based on tokenomics // International Journal of Financial Studies. 2024. Vol. 12. № 4. P. 107-110.
7. Drogunova Y. Integration of UI and API testing into CI/CD processes as a factor in accelerating the release of digital products // Universum: technical sciences: electron. scientific journal. 2025. № 5(134). P. 26-29.
8. Anchor Wallet for Desktop and Mobile / Greymass // URL: https://www.greymass.com/anchor (date of application 15.08. 2025).

UDC 004.89:65.011

PREDICTIVE ANALYTICS BASED ON MACHINE LEARNING AS A TOOL FOR COST OPTIMIZATION IN OPERATIONS MANAGEMENT

Mukayev T.
master's degree, Department of engineering mathematics and technology, University of Bristol (Bristol, United Kingdom)

ПРЕДИКТИВНАЯ АНАЛИТИКА НА ОСНОВЕ МАШИННОГО ОБУЧЕНИЯ КАК ИНСТРУМЕНТ ОПТИМИЗАЦИИ ЗАТРАТ В ОПЕРАЦИОННОМ УПРАВЛЕНИИ

Мукаев Т.М.
магистр, Факультет энергетики и нефтегазовой индустрии, Бристольский университет (Бристоль, Великобритания)

Abstract

This article examines the role of predictive analytics, based on machine learning methods, in enhancing the efficiency of operations management. It explores approaches to the application of predictive analytics in cost management, resource and supply chain management, as well as in maintenance planning. Particular attention is given to the resilience of business processes and the reduction of inefficiencies through accurate forecasting and the integration of analytical tools into strategic planning. It investigates the potential of predictive models to reduce costs, improve equipment reliability, and optimize supply chains.

Keywords: predictive analytics, machine learning, operations management, cost optimization, resource management, supply chain management.

Аннотация

В данной статье рассматривается роль предиктивной аналитики на основе методов машинного обучения в повышении эффективности операционного управления. Изучаются подходы к применению предиктивной аналитики в управлении затратами, ресурсами и снабжением, а также в планировании технического обслуживания. Особое внимание уделяется вопросам устойчивости бизнес-процессов и сокращения непроизводительных расходов за счет точных прогнозов и интеграции аналитических инструментов в стратегическое планирование. Исследуется потенциал использования предиктивных моделей для снижения издержек, повышения надежности работы оборудования и оптимизации цепочек поставок.

Ключевые слова: предиктивная аналитика, машинное обучение, операционное управление, оптимизация затрат, управление ресурсами, управление снабжением.

Introduction

In the course of constant competition and an unstable economic environment, companies are starting to seek to increase the efficiency of operations management. It is now possible through the incorporation of analytical tools. Generally established methods of planning and cost control often prove insufficiently flexible. They are primarily focused on examining historical data and do not adequately account for the dynamic changes in both internal and external conditions. Against this scenery, predictive analytics assumes particular importance.

Machine learning (ML) serves as the technological bases of predictive models, as it provides high forecasting accuracy and supports the automation of decision-making processes. The application of such approaches creates new opportunities for cost optimization, maintenance planning, and supply chain management. Evidence from practice often demonstrates that accurate forecasts not only contribute to cost reduction but also reduce possible risks associated with resource chain disruptions or equipment failures. The goal of this research is to examine predictive analytics based on ML as a tool for cost optimization in operations management.

Main part. Predictive analytics and ML in operations management

The spread of digital technologies has altered managerial approaches to process management in production. Predictive analytics has attracted special attention in the last few years as involving statistical techniques and ML algorithms to predict the future state of systems and processes. Unlike conventional analytical tools that focus primarily on retrospective data analysis, predictive methods aim to reveal patterns that can describe likely scenarios of future development (table 1).

Table 1

Traditional and predictive approaches to operations management [1, 2]

Aspect	Traditional approach	Predictive approach
Cost planning	Reactive (based on actuals). Manual estimation of expenses.	Proactive (based on forecasts). Data-driven optimization of budget allocation.
Supply management	Excessive inventories or shortages. Limited to recent usage metrics.	Optimized inventory levels. Uses historical trends, seasonality, and demand forecasts.
Maintenance	Scheduled maintenance or reactive repairs upon failure.	Predictive maintenance based on probability of failure and real-time equipment data.
Resource utilization	Static allocation. Based on estimated peak loads.	Dynamic adaptation. Aligned with forecasted demand across time intervals.
Cost efficiency	Prone to overprovisioning or underutilization.	Optimized for both cost efficiency and performance.

Predictive analytics occupies an intermediate position between descriptive and prescriptive analytics. While the former answers the question «what has happened? » and the latter addresses «what actions should be taken? », the predictive level provides insight into «what is likely to happen? ». In this sense, it is not merely a matter of recording past events, but of modeling possible future developments.

These edges can explain the expanding interest in predictive analytics across industries. As more and more business processes become digital, the need for tools that can make accurate predictions keeps spreading. Because of this, the predictive analytics market is experiencing stable growth, as depicted by the projected increase in revenue in the coming years (fig. 1).

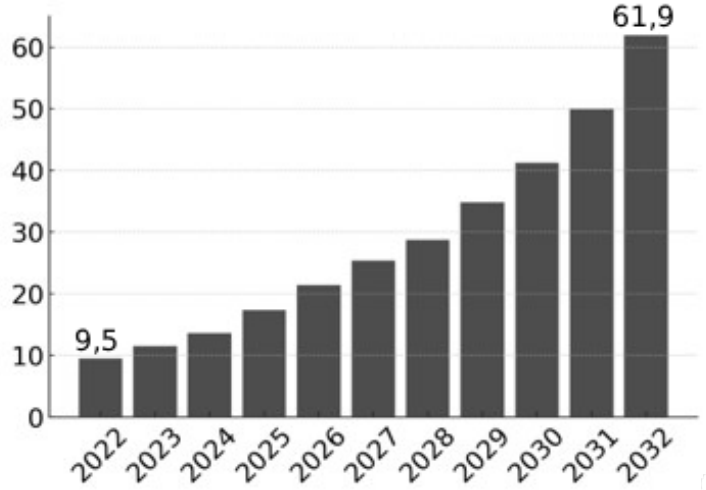


Figure 1. Predictive analytics market revenue forecast worldwide, billion dollars [3]

Algorithms based on ML serve as the fundamental technological basis of predictive analytics. This branch of artificial intelligence relies on algorithms that are trained on historical data and are capable of identifying complex nonlinear relationships that traditional statistical methods cannot capture [4]. Among the most widely used algorithms in operations management tasks are linear and logistic regression, decision trees, ensemble methods (such as gradient boosting and random forests), as well as neural networks.

Each of these approaches has distinct advantages. For instance, in retail sales forecasting, an optimized Random Forest algorithm achieved a coefficient of determination of $R^2 = 0.945$, compared to 0.531 for linear regression, while also reducing the root mean squared logarithmic error by 0.117. Neural networks, in turn, have proven highly effective for handling large and unstructured datasets [5]. Further benefits are provided by super-ensemble models, which can significantly improve forecasting accuracy. In streamflow forecasting, predictive accuracy increased by 20,6% compared to linear regression, while neural networks demonstrated improvements of 15-17% [6].

The application of ML in operations management enables a shift from static to dynamic planning. At the same time, the implementation of predictive models entails a number of challenges. Chief among them is the **issue of data quality**. Incomplete or imbalanced datasets can lead to significant forecasting errors. A telling example is **Unity,** where the ingestion of incorrect data from a major client into the ML algorithm used for ad placement not only slowed growth but also compromised the performance of the model. According to the company's management, the resulting losses in 2022 amounted to approximately $110 million [7].

Another challenge lies in the **interpretability of models**. The most accurate algorithms, such as deep neural networks, often function as a «black box», making it difficult to explain managerial decisions derived from their forecasts. On top of this problem, the issue of overfitting must be considered, wherein a model shows high accuracy on historical data but proves ineffective when predicting new situations.

Despite these circumspections, the potential of predictive analytics in the framework of operations management remains considerable. It helps managers to forecast system behavior and to identify the main factors influencing future changes. This knowledge serves as a starting point for developing cost-optimization strategies, making predictive technologies an integral component of modern management concepts.

In this way, predictive analytics and ML provide the foundation for transitioning to proactive operations management. Their use allows organizations to view costs and resources not as static entities but as dynamic variables, thereby creating the conditions for building resilient management models.

Cost optimization and resource management based on forecasts

Contemporary organizations strive to increase the efficiency of operational processes through precise planning and the minimization of uncertainty. Within the context of it, predictive analytics emerges as a tool that enables decision-making informed by the anticipated future states of systems and processes.

Forecasting costs plays a central role in operations management. Predictive models allow for the construction of development scenarios and the comparison of alternative budget allocation strategies. In the manufacturing sector, such models can anticipate increases in raw material costs due to market price fluctuations, thereby providing organizations to adjust procurement policies in advance.

In the service sector, forecasting supports the adjustment of personnel expenses in response to seasonal variations in demand. Thus, cost management evolves from being merely an accounting mechanism for recording expenditures into a dynamic system capable of adapting to changing conditions. Evidence from a study of over 30000 U.S. manufacturing establishments further demonstrates that firms actively employing predictive analytics achieve significantly higher productivity, with average sales approximately $918000 greater than those of comparable competitors [8].

One of the most promising directions is the use of predictive models in **financial risk management**. Through scenario modeling, it is possible to evaluate which specific events, for example, supply chain delays, rising energy tariffs, or equipment failures, may lead to significant budget deviations and to plan compensatory measures in advance.

Supply chain and **resource management** represent some of the most cost-intensive areas for any organization. Predictive data analysis makes it possible to generate precise demand projections and identify ideal inventory levels. Companies frequently either keep excessive stockpiles or experience shortages that interrupt production cycles under conventional management practices. Various ML models are used in supply chains and logistics to estimate delivery times and determine the likelihood of delays. The integration of supply chain management (SCM) systems with predictive analytics should receive special attention. Such an approach creates the conditions for more accurate procurement planning and a reduction in nonproductive expenditures [9].

An important area of application for prospective analysis is **predictive maintenance**. Traditional approaches involve scheduled servicing at predetermined intervals. This methodology is not always considered efficient, since equipment may fail earlier than expected or operate reliably for longer than planned. Predictive models make it possible to forecast the likelihood of failure based on data on equipment condition and external factors. This method facilitates substantial reductions in repair costs by preventing unplanned downtime and alleviating expenditures associated with unscheduled maintenance (fig. 2).

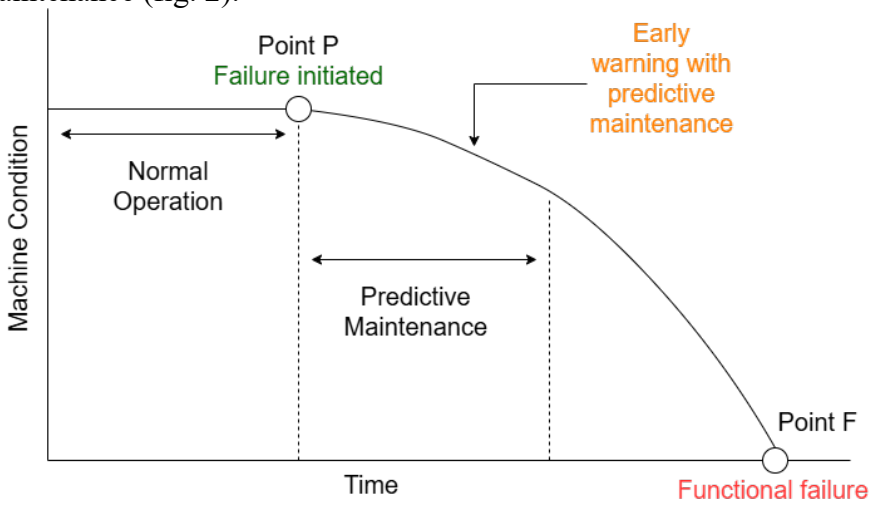


Figure 2. Potential failure diagram presenting inspection intervals and predictive maintenance [10]

Practical applications of predictive analytics demonstrate significant economic benefits. Empirical findings from a 2025 study further indicate that the use of predictive analytics across organizations in various sectors has a notable impact on key dimensions of performance. Respondents reported improvements in forecast accuracy, acceleration of decision-making processes, reductions in operational costs, and higher levels of customer satisfaction (table 2).

Table 2

Impact of predictive analytics on business performance [11]

Metric	Mean (%)	Standard deviation
Predictive analytics adoption	67,4	10,6
Decision-making speed improvement	73,3	10,2
Forecasting accuracy	76,6	10,2
Operational cost reduction	60,4	12,1
Customer satisfaction increase	67,6	10,1

In addition to examples of predictive analytics found in academic research, there are also notable cases of its application in industry. For instance, Pinterest achieved a 20% reduction in infrastructure costs through more accurate cloud resource scaling, ensuring payment only for the capacity actually utilized. Similarly, Slack Technologies reduced cloud service expenses by 15-20% by detecting billing anomalies early and renegotiating contracts with providers [12].

McDonald's, as part of its digital transformation and in collaboration with Google Cloud, integrated predictive analytics into procurement and inventory management processes across more than 40000 restaurants. This initiative helped the company to optimize supply chains, lower costs, and boost the resilience of its supply operations [13].

A common thread across these cases is the transition to a new mode of cost management. Rather than reacting to past events, organizations are acting proactively. This shift reduces expenditures and fosters the development of long-term sustainability strategies, where each forecast becomes a tool for strengthening competitiveness [14]. Leveraging predictive insights in finance, procurement, resource management, and maintenance thus lays the groundwork for more resilient business models.

Conclusion

Predictive analytics powered by ML is turning into a valuable asset in terms of improving the effectiveness in operation management. Through the use of forecasting models, entities are in a position to plan for future states in their systems, check prospective risks, as well as arrive at more prudent resource and cost allocation decisions. Through this, entities are in a position to transition from reactive management approaches to proactive management approaches whose core is anticipating potential modifications as well as their abilities to change ahead of time.

The use of predictive technologies in supply chain management and budgetary planning shows great promise for lessening expense while guaranteeing the durability of business processes. The integration of precise forecasts with adaptive management approaches serves to reduce inefficiencies to a minimum, enhance equipment reliability, and better-optimize supply chain systems. Aligned together, these variables put predictive analytics on solid ground as the means to establishing sustainable long-term sources of competitiveness.

References

1. Yarov Y. Optimization of business processes in construction companies using digital technologies and automation // Sciences of Europe. 2025. № 167. P. 67-70.
2. Zharmagambetov Y. Application of machine learning algorithms in financial risk management systems // International Research Journal of Modernization in Engineering Technology and Science. 2025. Vol. 7. № 5. P. 1503-1509.
3. Predictive Analytics Statistics 2025 By A Practical Approach / Market.us Scoop / URL: https://scoop.market.us/predictive-analytics-statistics/ (date of application: 17.08.2025).
4. Kiselev R. Cyberattack prediction models using machine learning // Professional Bulletin: Information Technology and Security. 2024. № 1/2024. P. 24-28.
5. Ganguly P., Mukherjee I. Enhancing retail sales forecasting with optimized machine learning models // In2024 4th International Conference on Sustainable Expert Systems (ICSES). 2024. P. 884-889.
6. Hristos T., Georgia P., Andreas L. Super ensemble learning for daily streamflow forecasting: large-scale demonstration and comparison with multiple machine learning algorithms // Neural Computing & Applications. 2021. Vol. 33(8). P. 3053-3068.
7. The Impact of Bad Data and Why Observability is Now Imperative / IBM / URL: https://www.ibm.com/think/insights/observability-data-benefits (date of application: 23.08.2025).
8. Brynjolfsson E., Jin W., McElheran K. The power of prediction: predictive analytics, workplace complements, and business performance // Business Economics. 2021. Vol. 56(4). P. 217-39.
9. Stepanov M. Adaptive control systems for optimizing electric drive operation and reducing energy consumption in challenging conditions // Original research. 2024. Vol. 14. № 9. P. 86-92.
10. Achouch M., Dimitrova M., Ziane K., Sattarpanah Karganroudi S., Dhouib R., Ibrahim H., Adda M. On Predictive Maintenance in Industry 4.0: Overview, Models, and Challenges // Applied Sciences. 2022. Vol. 12(16). № 8081. P. 1-22.

11. Alhumaidi N. Quantitative Analysis of Predictive Business Analytics for Dynamic Decision-Making: A Survey-Based Study on Organizational Strategy Optimization // International Business & Economics Studies. 2025. Vol. 7. P. 103-124.

12. Kass E. Predictive Analytics for Cloud Resource Planning and Cost Forecasting / Authorea // URL: https://www.authorea.com/users/925546/articles/1297287-predictive-analytics-for-cloud-resource-planning-and-cost-forecasting (date of application: 27.08.2025).

13. Kotagi V. Leveraging Big Data and Business Intelligence: A Case Study of McDonald's Competitive Advantage // Research and Applications of Web Development and Design. 2024. Vol. 8(1). P. 8-14.

14. Nazarova Ye. The influence of psychoanalytic practices on leadership and organizational culture // International Journal of Professional Science. 2025. № 4(1). P. 71-77.

UDC 004.8: 519.6

FEATURE SELECTION METHODS IN MACHINE LEARNING: FROM SIMPLE FILTERS TO INTERPRETABILITY WITH SHAP

Bondarenko K.
master's degree, HSE University (Moscow, Russia)

МЕТОДЫ ОТБОРА ПРИЗНАКОВ В МАШИННОМ ОБУЧЕНИИ: ОТ ПРОСТЫХ ФИЛЬТРОВ ДО ИНТЕРПРЕТИРУЕМОСТИ С SHAP

Бондаренко К.А.
магистр, Национальный исследовательский университет «Высшая школа экономики» (Москва, Россия)

Abstract

This paper examines feature selection methods in machine learning tasks and their impact on model performance and interpretability. The problem of high-dimensional data remains one of the most critical challenges in modern analytics, as redundant and irrelevant features increase the risk of overfitting, complicate computations, and reduce the transparency of conclusions. The aim of this study is to provide a comparative analysis of filter, wrapper, and embedded feature selection methods, as well as interpretation techniques based on SHAP, with an emphasis on their practical application. As an example, the widely used California Housing dataset was employed for modeling and feature importance evaluation. The analysis utilized permutation importance, partial dependence plots, and SHAP to assess and compare the relevance of features.

Keywords: Feature selection, filter methods, wrapper methods, embedded methods, interpretability, SHapley Additive exPlanations (SHAP).

Аннотация

В данной работе рассматриваются методы отбора признаков в задачах машинного обучения и их влияние на качество моделей и интерпретируемость результатов. Проблема высокой размерности данных остаётся одной из важнейших в современной аналитике, поскольку избыточные и нерелевантные признаки увеличивают риск переобучения, усложняют вычисления и снижают прозрачность выводов. Цель исследования заключается в сравнительном анализе фильтрационных, wrapper- и embedded-методов отбора признаков, а также интерпретационных техник на основе SHAP, с акцентом на их практическое применение. В качестве примера использован популярный датасет California Housing, на котором проведено моделирование и оценка важности признаков. Для анализа применялись методы permutation importance, partial dependence plots и SHAP-анализ.

Ключевые слова: отбор признаков, фильтрационные методы, wrapper-методы, embedded-методы, интерпретируемость, SHapley Additive exPlanations (SHAP).

Introduction

In machine learning (ML), features are measurable characteristics of an object that serve as the basis for model predictions. For example, in housing price prediction tasks, features may include floor area, number of rooms, year of construction, and location. The choice and representation of features directly influence the effectiveness of the model, including its accuracy, robustness to noise, and generalization ability.

Statistical reports indicate a steady increase in both the number of data sources and the volume of available information [1]. These datasets vary in complexity and interpretability, which can

become a challenge for model development. Modern datasets may contain hundreds or even thousands of features, many of which can be irrelevant, redundant, or even detrimental to training. Consequently, feature selection remains a major step in data preprocessing, as it helps improve certain metrics and speed up model performance across a wide range of applications, particularly in medicine and finance. The importance of feature selection is further reinforced by growing requirements for explainable artificial intelligence (XAI) [2]. Stakeholders increasingly demand to understand how and why a model arrives at specific outcomes. Methods such as SHAP provide post hoc explanations, increasing trust among users, clients, and regulatory bodies.

Main part. Feature selection in ML

High-dimensional datasets are occurring more often in modern ML applications, ranging from biomedical records and sensor data to financial transactions and industrial monitoring. While the availability of numerous features potentially expands the descriptive power of models, it also introduces various technological and methodological challenges. Irrelevant variables can increase computational cost, complicate model interpretation, and lead to overfitting. Adequate feature selection addresses these issues by identifying and retaining the most informative subset of variables. Practical studies show that eliminating irrelevant variables and focusing on the most significant features can lead to a substantial increase in accuracy, up to 15% compared to models that use all available data [3].

In general, feature selection methods aim to balance three objectives: efficiency (reducing the dimensionality of data to speed up training and inference), accuracy (improving generalization by removing noise), and interpretability (allowing domain experts to understand which variables play a key role in decision-making). Unlike dimensionality reduction techniques such as principal component analysis (PCA), which transform the feature space into new components, feature selection preserves the original variables, making the results easier to communicate in applied domains.

Three broad families of feature selection techniques are typically distinguished: filter, wrapper, and embedded methods. Each of these approaches reflects a trade-off between computational efficiency, predictive performance, and interpretability. In practice, the choice of method can depend on the dataset size, model complexity, and the ultimate purpose of the analysis.

Filter methods

Filter methods are popular due to their simplicity and efficiency at the initial stage of data analysis. According to recent studies [4], they are among the most widely used approaches in ML feature selection research. These methods are independent of a specific model algorithm and rely only on the statistical relationship of each feature with the target variable or on the general properties of the data (fig. 1).

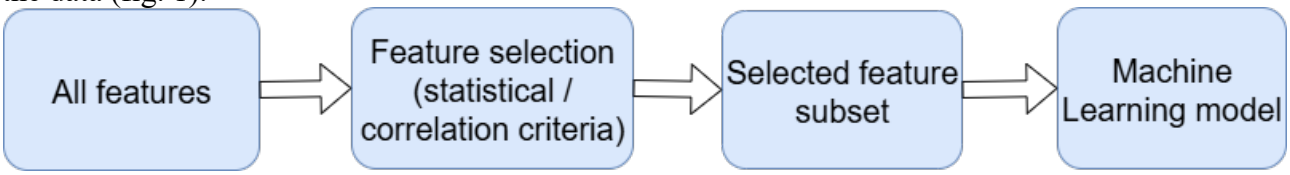


Figure 1. Diagram of filter methods

One of the common approaches is **correlation analysis**, which involves studying correlations between features and the target variable, as well as correlations among the features themselves. The logic is that informative features should be strongly correlated with the target variable, while at the same time features should not be highly correlated with each other, so as not to duplicate the same information. In practice, this is implemented by calculating correlations (e.g., Pearson's coefficient for pairs of numerical variables) and removing features that are either weakly related to the target or strongly duplicating other features.

Another widely used technique is **Chi-square test (χ^2)**. For classification problems with categorical (discrete) features, the χ^2 test is commonly applied to assess the statistical dependence between a feature and the target class. The χ^2 statistic is computed between the distribution of feature values and classes, and the result is compared with the expected value under the null hypothesis of independence. Features are ranked according to their χ^2 values (or the corresponding p-values). A

high χ^2 value and low p-value suggest that the feature is likely dependent on the target class, making it informative for classification.

For feature selection in classification tasks with numerical variables, the **analysis of variance (ANOVA) F-test** is widely used. The one-way ANOVA F-test compares the variance of feature values between groups (classes) with the variance within groups. A high F-statistic indicates that the mean values of the feature differ significantly across classes compared to within-class variation. Features with high F-values are therefore considered discriminative and informative. In ML libraries (e.g., **sklearn.feature_selection**), ANOVA is implemented as the **f_classif** method. It returns an F-value and corresponding p-value for each feature, which can be used to rank and select the top-N features. However, ANOVA assumes approximate normality of distributions and homogeneity of variances, so results should be interpreted with caution for skewed or heteroscedastic data.

Finally, another powerful criterion is **mutual information**, which measures the amount of shared information (dependency) between a feature and the target variable. Mutual information estimates how much the uncertainty of the target variable is reduced when the feature is known. Formally, it quantifies the number of bits of information the feature contains about the class [5]. A key advantage of mutual information is its ability to capture non-linear and non-parametric dependencies between X and Y, unlike correlation which is limited to linear relationships. Features with high mutual information relative to the target are considered highly relevant. In practice, mutual information can be estimated for each feature via discretization or kernel density estimation. In scikit-learn, the functions **mutual_info_classif** and **mutual_info_regression** are available for this purpose.

Filter methods remain a popular choice at the initial stage of data analysis due to their simplicity and model-agnostic nature. However, their main limitation lies in the fact that they evaluate features independently of the chosen learning algorithm. As a result, the selected subset of features may not always align with the characteristics of a specific model. Therefore, practitioners may resort to other feature selection strategies.

Wrapper methods

Wrapper methods perform feature selection by evaluating the performance of a model on different subsets of features. Unlike filter methods, which assess the importance of features independently of the model, wrapper approaches “wrap” the selection process around a ML algorithm, using its performance metric to determine the usefulness of features. In this case, candidate subsets are generated, evaluated with a model, and iteratively refined until the best-performing subset is selected (fig. 2).

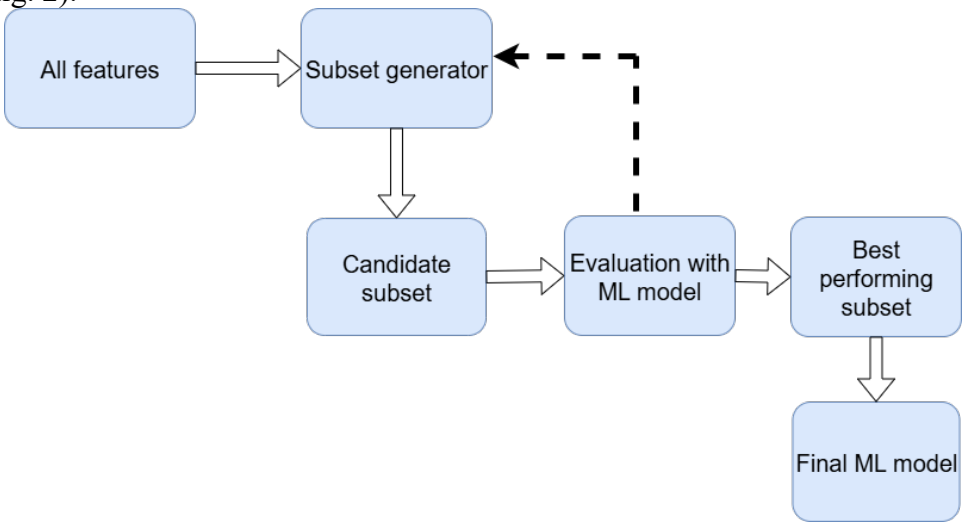


Figure 2. Diagram of wrapper methods

This approach allows capturing nonlinear relationships and interactions between features, for example, combinations of features that individually have little impact on the result but together significantly improve the model. The main drawback of wrapper methods is their high computational cost, since the model must be trained repeatedly on different feature sets. To avoid overfitting during the selection process, cross-validation is usually applied to evaluate the model at each step.

One of the main approaches is **Recursive Feature Elimination (RFE)**, which iteratively eliminates less important features by repeatedly training the model. It starts with the full set of features and fits a base algorithm (e.g., decision tree or logistic regression). The least significant feature (based on the model's importance estimate, such as coefficient value or contribution to performance metric) is then removed, and the model is retrained on the reduced set [6]. This process continues recursively, eliminating one or even several features at each step, until the desired number of most important features remains. In this way, RFE implements a "greedy" backward elimination strategy, similar to stepwise regression. The advantage of RFE is that it considers the effect of features in the context of the model, allowing it to identify an optimal subset even when features are correlated. However, for datasets with very high dimensionality, RFE can be computationally expensive.

Another important method is **Sequential Feature Selection (SFS),** which represents a family of methods where features are added or removed one at a time in a sequence of steps, with the model performance evaluated at each stage. SFS can operate in two modes: stepwise forward selection or stepwise backward elimination. In the **forward** mode, the algorithm starts with an empty set. On the first step, all features are evaluated individually, and the one that gives the best model performance (e.g., highest cross-validated accuracy) is selected. On subsequent iterations, each remaining candidate feature is tested in combination with the already selected set, and the one that provides the greatest performance gain is added. The process continues until a predefined number of features is reached or no further performance improvement is observed. In the **backward** mode, the algorithm starts with the full feature set and iteratively removes the least useful feature, evaluating model performance without each candidate, until the stopping criterion is met.

In summary, wrapper methods allow for the identification of feature subsets optimized for a specific model, often leading to superior predictive performance. However, their high computational demands and susceptibility to overfitting make them less practical for very large datasets. This trade-off has led to increasing interest in embedded methods, which integrate feature selection directly into the model training process.

Embedded methods

Embedded methods combine the process of feature selection with model training, meaning that selection occurs "internally" during the algorithm's learning phase. These approaches are considered a compromise between simple filter methods and computationally expensive wrapper methods [7]. On the one hand, the model itself "decides" which features are important (as in wrapper methods), while on the other hand there is no need to manually iterate through multiple models, since the selection is computationally more efficient. In fact, the learning algorithm implicitly performs feature selection by regularizing or evaluating the contribution of features (fig. 3).

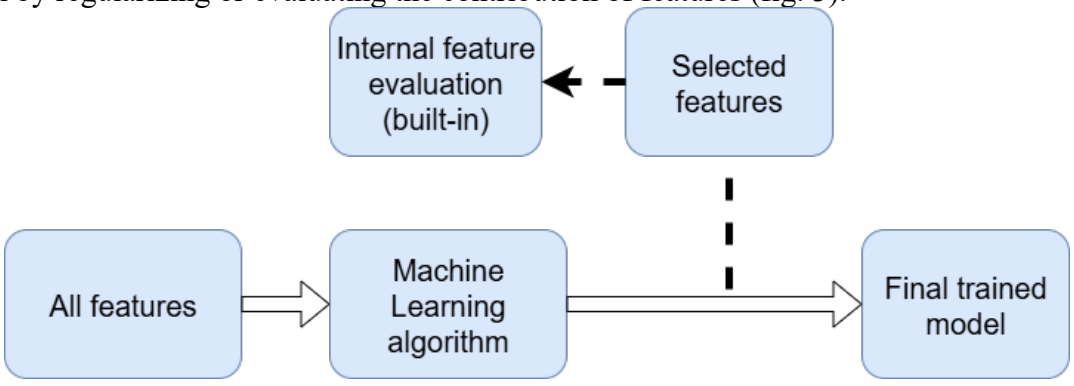


Figure 3. Diagram of embedded methods

At a preliminary stage, all features are provided to the learning algorithm, which then evaluates their relevance internally during training. As a result, only the most significant predictors are retained and directly used in the final model. This integration of training and feature selection shows the main advantage of embedded approaches – features are evaluated as part of model building, not in a separate step.

L1 Regularization (LASSO). Regularization with the L1 norm (e.g., in the linear Lasso model) performs feature selection by adding a penalty term to the loss function proportional to the sum of

the absolute values of the model coefficients. As a result of this penalty, many coefficients are shrunk exactly to zero, the corresponding features are automatically excluded from the model. Thus, L1 regularization simultaneously improves the generalization ability of the model (by preventing overfitting through complexity control) and performs embedded feature selection by setting irrelevant parameters to zero. By contrast, L2 regularization (Ridge) reduces coefficients but does not drive them to zero, so L2 does not actually perform feature selection. A practical illustration of the effectiveness of L1 regularization can be found in studies using the EEG (Electroencephalography) Emotion dataset, where combining Random Forest (RF) with LASSO feature selection improved classification accuracy from 98.78% to 99.39%, demonstrating how penalization and embedded feature selection can increaase model performance in high-dimensional biomedical data [8].

Decision Trees and Ensembles (Random Forest, XGBoost, LightGBM, etc.). Tree-based algorithms have a built-in ability to evaluate feature importance through specific importance measures. During tree construction, features that provide the greatest reduction in impurity or error are chosen for splits. After training, it is possible to compute how much each feature reduced the splitting criterion on average, this serves as the feature importance metric. In RF (ensembles of many trees), importances are averaged across all trees. In gradient boosting, the contributions of each feature are accumulated over all base learners. Importances are typically normalized so that their sum across all features equals 1.

Embedded approaches combine model training and feature selection within a single process. Their advantage lies in the ability to directly exploit the structure of the learning algorithm, yielding feature subsets that are optimized for the chosen model. However, the results are often model-dependent, meaning that different algorithms may assign different levels of importance to the same features.

Interpretability and explanation with SHAP

SHAP (SHapley Additive exPlanations) is a modern method for interpreting the results of ML models, based on the concept of Shapley values from game theory. Unlike feature selection methods, which aim to identify significant features for the model, SHAP is used for *post hoc* explanation of an already trained model. It decomposes the model's prediction into the sum of individual feature contributions, showing how much and in what direction each feature influenced a specific prediction. The sum of all SHAP values for an observation, plus the baseline value, equals the model's prediction, ensuring local accuracy of the explanation.

One of the key strengths of SHAP is its model-agnostic nature: it can be applied to any ML model as an interpretation tool without requiring modifications to the algorithm itself. Optimized implementations exist for decision trees and tree-based ensembles, making SHAP efficient enough to compute even on large datasets. SHAP provides both local explanations (at the level of individual predictions) and global interpretability (overall feature importance across the model).

With its help, one can ensure that the model relies on expected, meaningful features rather than on random noise factors. Moreover, SHAP results can help identify problematic dependencies. If SHAP reveals a suspiciously strong influence of a non-obvious feature, this may indicate data leakage or correlations requiring further analyst attention.

To compute SHAP values, the algorithm considers all possible feature coalitions and the marginal contribution of each feature to improving the prediction when it is added to a coalition. Exhaustively enumerating all combinations of features is computationally expensive, so approximation algorithms have been developed. In particular, the authors of SHAP proposed specialized methods to accelerate computation: the model-agnostic **KernelSHAP** and the high-performance **TreeSHAP** for decision tree models [9].

In addition to Shapley-value-based methods (SHAP), another widely used technique for interpreting ML models is **LIME (Local Interpretable Model-agnostic Explanations)**. Its key idea is to approximate the behavior of a complex "black-box" model in the vicinity of a single observation using a simple and interpretable model, such as linear regression. This approach makes it possible to explain individual predictions by highlighting the features that had the strongest influence on the outcome for that specific instance.

The advantage of LIME lies in its simplicity and universality. It can be applied to any model and type of data with relatively low computational costs. However, interpretations obtained through LIME may be unstable. Small changes in the data or parameters can lead to significant differences in explanations. For this reason, LIME is often regarded as a tool for rapid prototyping or preliminary hypothesis testing, whereas SHAP is used for more rigorous and reproducible analysis (table 1).

Table 1

SHAP and LIME differences [10, 11]

Aspect	SHAP	LIME
Theoretical basis	Based on Shapley values from cooperative game theory	Based on local linear approximations of the model
Output consistency	Provides consistent and additive explanations	May produce different explanations for the same instance depending on sampling
Scope	Both local (per-instance) and global (overall feature importance) explanations	Primarily local explanations for individual predictions
Model coverage	Model-agnostic, with optimized implementations for tree-based models	Fully model-agnostic, works with any black-box model
Computation cost	Higher, especially for complex models, though TreeSHAP reduces cost	Generally faster, but depends on number of samples used
Interpretability	Quantifies exact contribution of each feature to prediction	Provides approximate influence of features using surrogate model
Stability	More stable and reproducible explanations	More sensitive to randomness and sampling variations

SHAP provides more reliable and reproducible explanations, allowing not only the identification of the most important features but also the visualization of their influence in an intuitive way. Such tools make ML results understandable and accessible even to specialists without deep expertise in data science.

Practical study

To illustrate the application of interpretation techniques and feature importance evaluation, an experimental modeling study was conducted. The experiments were conducted in Python 3.11 (Jupyter Notebook, Anaconda) using the libraries scikit-learn v1.5, SHAP v0.44, and matplotlib v3.9. As a test dataset, the California Housing dataset from scikit-learn was used (20,640 observations, 8 features). The models were built with the Random Forest Regressor algorithm configured with 400 trees and the parameters max_depth=None, min_samples_split=2, and min_samples_leaf=1. Feature importance was analyzed using three methods: Permutation Importance, Partial Dependence Plot, and SHAP values.

At the initial stage of analysis, it is often reasonable to apply traditional statistical methods in order to evaluate straightforward dependencies between features. Such techniques provide a preliminary understanding of the dataset and can highlight apparent correlations before more advanced modeling is applied. The correlation heatmap in this case illustrates linear relationships among the housing attributes, showing, in particular, a strong positive correlation between AveRooms and AveBedrms, as well as a link between MedInc and the target variable MedHouseVal (fig. 4).

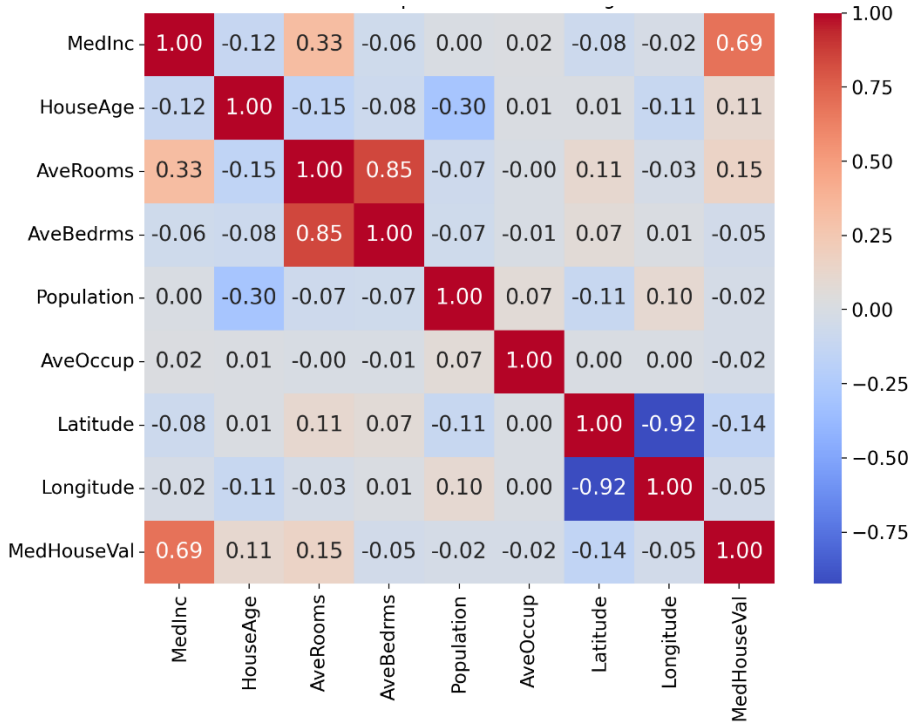


Figure 4. Correlation heatmap of the California housing dataset

Nevertheless, correlation analysis is limited to detecting linear dependencies and does not account for non-linear or interaction effects that may play a significant role in predictive modeling. For this reason, subsequent sections focus on ML-based approaches combined with interpretability techniques, which allow for a more complete assessment of feature relevance and their contribution to the predictive process. As a first step, feature importance was evaluated, since, as noted earlier, this measure is inherently embedded in the RF model (fig. 5).

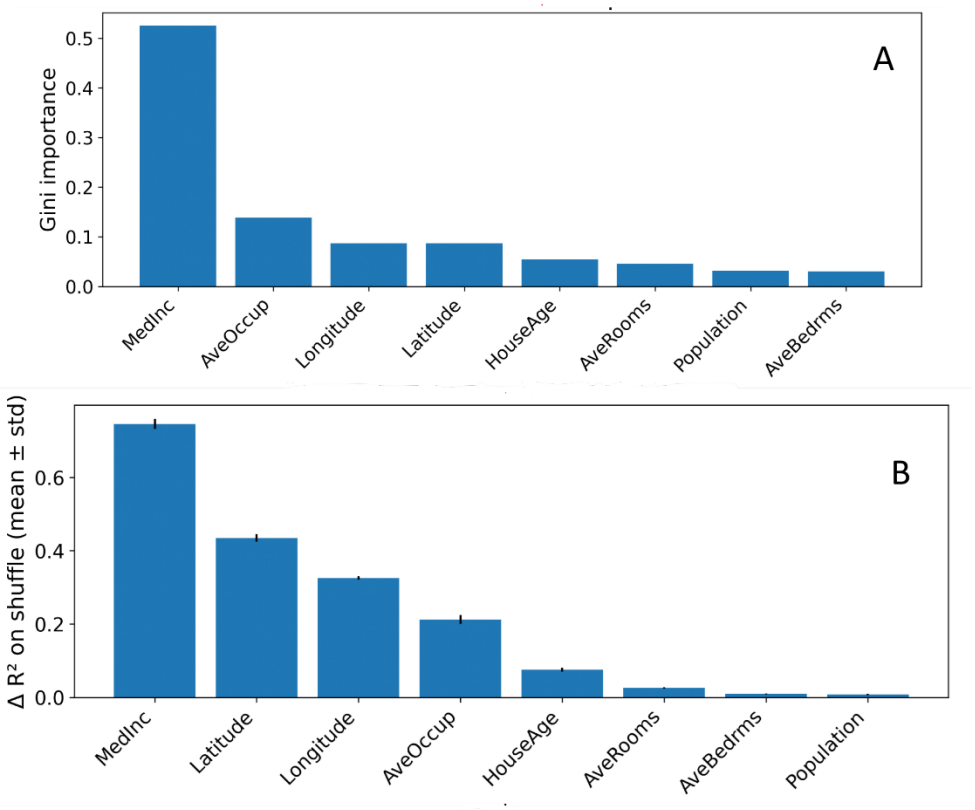


Figure 5. Feature importance in the California housing dataset: (A) Gini importance from the RF model, (B) Permutation importance on the validation set

Both Gini importance and permutation importance provide consistent results in identifying the most relevant features. This similarity is expected, since both approaches capture the degree to which a feature contributes to model performance. However, permutation importance is generally considered more robust, as it directly quantifies the change in predictive accuracy when feature information is disrupted, whereas Gini importance reflects the feature's role in reducing node impurity during tree construction.

To further investigate the marginal effect of individual predictors on the target variable, Partial Dependence Plots were constructed. These plots illustrate the average change in the predicted response as a given feature varies across its range, while all other features are marginalized (fig. 6).

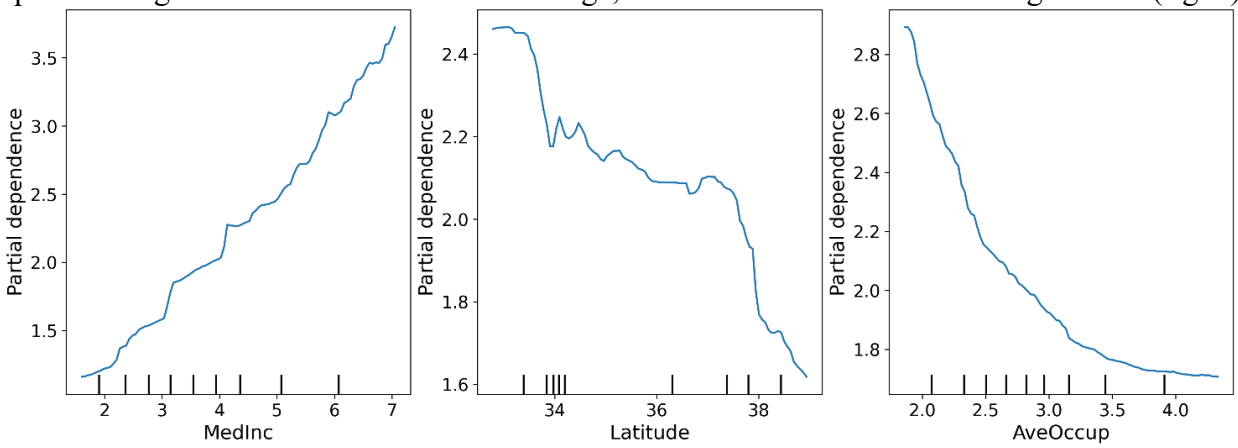


Figure 6. Partial Dependence Plots for the top-3 features

In particular, the these for MedInc, Latitude, and AveOccup demonstrate the most pronounced partial dependencies, consistent with their high importance in the model. Unlike feature importance measures, which quantify global relevance, PDP provide a visual representation of the functional form of the relationship between predictors and the target.

The SHAP analysis of the California Housing dataset highlights several key patterns in feature importance. The most influential factor is median income (MedInc), which strongly dominates the model's predictions and was also consistently identified as the primary driver of house value in earlier analyses. Geographical variables such as latitude and longitude also hold an important place, reflecting regional disparities in the California housing market. The bar plot of SHAP values presents the average magnitude of feature contributions across all predictions. This representation allows for ranking variables by their overall importance, providing a global view of the most influential predictors in the model (fig. 7).

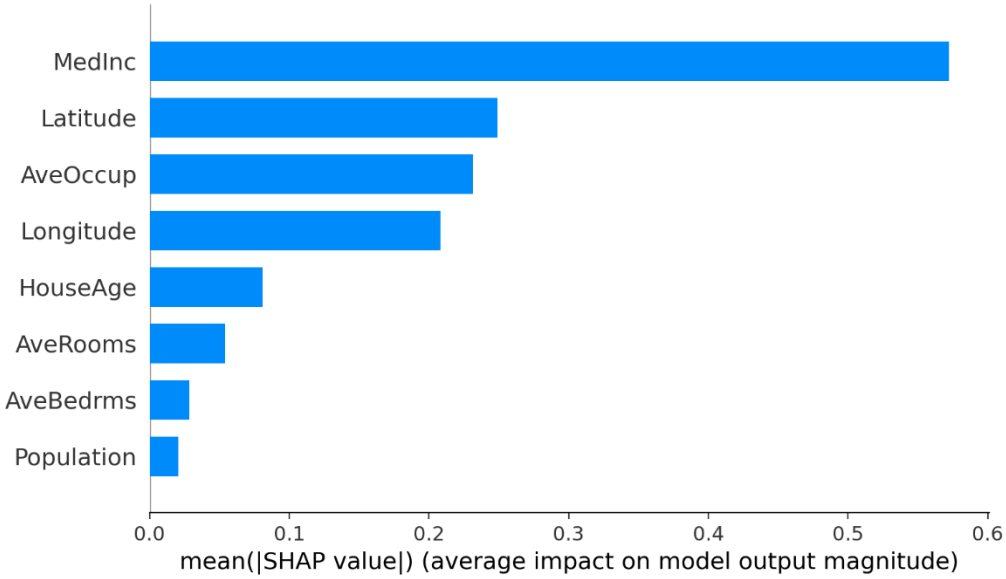


Figure 7. SHAP bar plot of mean absolute feature importance

The ranking of features based on SHAP values closely aligns with both Gini and permutation importance, which indicates methodological consistency across approaches. This reinforces that the observed structure is not an artifact of a single metric, but a robust outcome of the model training process. The dependence plot illustrates the relationship between SHAP values for the top-ranked feature and its actual values. This visualization shows how the contribution of the feature changes across its range, while also capturing interaction effects with other predictors, represented here by color coding (fig. 8).

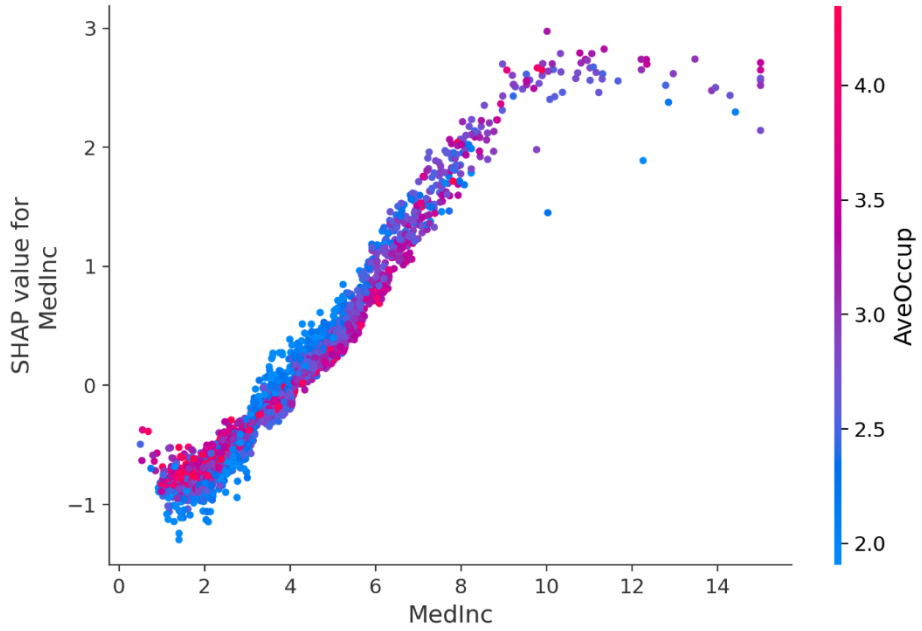


Figure 8. SHAP dependence plot for MedInc with AveOccup as interaction feature

Summary plot reveals not only the relative importance of predictors but also the direction and variability of their impact. It combines both global and local perspectives by displaying the distribution of SHAP values for each feature across all samples. In this case, the dependence plot demonstrates how SHAP values for MedInc vary across its range, indicating the feature's contribution to model predictions. The color gradient represents values of AveOccup, which allows the identification of potential interaction effects between features. This visualization thus provides both a univariate and a bivariate perspective, showing the main effect of MedInc while simultaneously spotlighting its relationship with AveOccup (fig. 9).

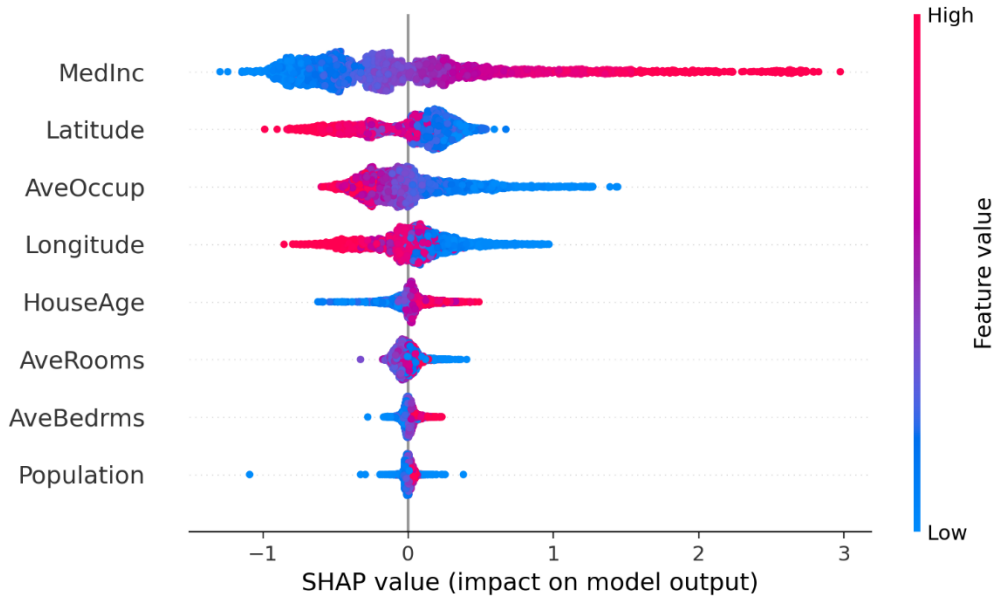


Figure 9. SHAP summary plot

Overall, the applied methods consistently highlighted a set of key predictors, such as median income, geographic coordinates, and average occupancy, which were repeatedly identified as having a substantial impact on the model output. This convergence across feature importance, permutation analysis, partial dependence plots, and SHAP values confirms the robustness of the results. It should be emphasized that the California housing dataset is a well-prepared and relatively clean benchmark dataset, whereas in real-world applications data often exhibit noise, missing values, multicollinearity, or complex nonlinear interactions. Such challenges typically complicate both model training and subsequent interpretation. In practice, data scientists and ML-engineers are often required to carefully choose appropriate methods or combine multiple approaches to effectively tackle these problems and ensure reliable feature selection and interpretation.

Nevertheless, SHAP-based interpretation offers a more overarching view of the underlying relationships, capturing both global patterns and local contributions of individual observations. For specialists, this means an easier way to communicate results to stakeholders who may not be fully familiar with the technical aspects of statistical modeling, ML algorithms, or the mathematics behind feature importance measures, but who are primarily interested in understanding the drivers of the observed outcomes and their practical implications.

Conclusion

Skilled feature selection remains an important component of ML, addressing vital problems of redundancy, overfitting, and interpretability that arise in high-dimensional datasets. Different approaches, including filter, wrapper, and embedded methods, provide complementary perspectives, balancing accuracy, and computational cost. Beyond improving predictive performance, feature selection contributes to the transparency and reliability of models, which are a major issue in domains requiring explainability.

In this study, the process of feature selection and interpretation was shown to illustrate how different methods underline the importance of individual predictors and how these insights can be presented in practice. Modern interpretability techniques, such as SHAP, extend these benefits by offering both global and local insights into model behavior. Together with tools like permutation importance and partial dependence plots, they form a wide framework for understanding the role of individual features in complex models. In practice, combining multiple techniques provides a more nuanced and trustworthy view, supporting technical optimization and informed decision-making.

References

1. Volume of data/information created, captured, copied, and consumed worldwide from 2010 to 2023, with forecasts from 2024 to 2028 / Statista / URL: https://www.statista.com/statistics/871513/worldwide-data-created/ (date of application: 10.08.2025).

2. Ali S., Abuhmed T., El-Sappagh S., Muhammad K., Alonso-Moral J.M., Confalonieri R., Guidotti R., Del Ser J., Díaz-Rodríguez N., Herrera F. Explainable Artificial Intelligence (XAI): What we know and what is left to attain Trustworthy Artificial Intelligence // Information fusion. 2023. Vol. 99. № 101805.

3. Cheng X. A Comprehensive Study of Feature Selection Techniques in Machine Learning Models / SSRN / URL: https://ssrn.com/abstract=5154947 (date of application: 14.08.2025).

4. Liyew C.M., Ferraris S., Di Nardo E., Meo R. A review of feature selection methods for actual evapotranspiration prediction // Artificial Intelligence Review. 2025. Vol. 58(10). № 292.

5. Zhou H., Wang X., Zhu R. Feature selection based on mutual information with correlation coefficient // Applied intelligence. 2022. Vol. 52(5). P. 5457-74.

6. Awad M., Fraihat S. Recursive feature elimination with cross-validation with decision tree: Feature selection method for machine learning-based intrusion detection systems // Journal of Sensor and Actuator Networks. 2023. Vol. 12(5). № 67.

7. Jiménez-Cordero A., Morales J.M., Pineda S. A novel embedded min-max approach for feature selection in nonlinear support vector machine classification // European Journal of Operational Research. 2021. Vol. 293(1). P. 24-35.

8. Abdumalikov S., Kim J., Yoon Y. Performance Analysis and Improvement of Machine Learning with Various Feature Selection Methods for EEG-Based Emotion Classification // Appl. Sci. 2024. Vol. 14. № 10511.

9. Aydoğan B., Aytekin T. An in-depth analysis of KernelSHAP and SamplingSHAP: assessing robustness, error, and efficiency: B. Aydoğan, T. Aytekin // Knowledge and Information Systems. 2025. P. 1-35.

10. Hasan M.M. Understanding model predictions: a comparative analysis of SHAP and LIME on various ML algorithms // Journal of Scientific and Technological Research. 2023. Vol. 5(1). P. 17-26.

11. Salih A.M., Raisi-Estabragh Z., Galazzo I.B., Radeva P., Petersen S.E., Lekadir K., Menegaz G. A perspective on explainable artificial intelligence methods: SHAP and LIME // Advanced Intelligent Systems. 2025. Vol. 7(1). № 2400304.

UDC 004.738.5:004.421.2

THE EVOLUTION OF WEB CRAWLING IN SEARCH ENGINES: PERFORMANCE, SCHEDULING, AND URL PRIORITIZATION

Bogutskii A.
bachelor's degree, ITMO University (St. Petersburg, Russia)

ЭВОЛЮЦИЯ ВЕБ-КРАУЛИНГА В ПОИСКОВЫХ СИСТЕМАХ: ПРОИЗВОДИТЕЛЬНОСТЬ, ПЛАНИРОВАНИЕ И ПРИОРИТЕЗАЦИЯ URL

Богуцкий А.Д.
бакалавр, Национальный исследовательский университет ИТМО (Санкт-Петербург, Россия)

Abstract

This article explores the evolution of web crawling systems used in modern search platforms. It examines key stages in the development of crawler architectures, ranging from simple sequential traversal to large-scale distributed systems. It analyzes various crawl scheduling methods, including URL loading strategies, queuing mechanisms, content change prediction, and adherence to web resource access constraints. Particular attention is given to URL prioritization as a critical factor influencing index completeness, the timely inclusion of new pages, and the overall effectiveness of search results. It also discusses machine learning–based approaches used to predict the importance of web pages and to improve the efficiency of crawler resource allocation.

Keywords: web crawling, search engine, URL traversal, URL prioritization, crawl scheduling, crawl strategies, resource allocation.

Аннотация

В данной статье исследуется эволюция систем веб-краулинга, применяемых в современных поисковых платформах. Рассматриваются ключевые этапы развития архитектур краулеров, начиная от простых последовательных алгоритмов обхода и заканчивая масштабируемыми распределенными системами. Анализируются методы планирования обхода, включая стратегии загрузки, очередности URL, прогнозирование изменений контента и соблюдение ограничений, накладываемых веб-ресурсами. Особое внимание уделяется приоритезации URL как фактору, влияющему на полноту индексации, своевременность включения новых страниц и общую эффективность поисковой выдачи. Также рассматриваются подходы на основе машинного обучения, используемые для предсказания значимости страниц и повышения эффективности распределения ресурсов краулера.

Ключевые слова: веб-краулинг, поисковая система, обход URL, приоритезация URL, планирование загрузки, стратегии обхода, распределение ресурсов.

Introduction

The creation of search engines is closely tied to approaches enabling the automated extraction of information from the internet. Web crawlers constitute an important element within the approach whose role is the retrieval, processing, and distribution of information to users. The constant growth of web pages and the increased structural complexity of them constitute important problems, emphasizing both the technological challenges as well as the research importance of web crawling.

Current crawlers operate under a wide range of constraints. The overall adoption of anti-bot protections and the emergence of partially restricted web segments compel crawlers to make more informed decisions about which resources to access. They must determine the optimal frequency of revisits and allocate computational resources efficiently.

Even though there have been big improvements in the field, there are still many dilemmas that need to be dealt with. These hurdles may consist of making page value prediction more accurate, taking into account how web content changes over time, or even adjusting to how users behave. Also, the strategies used during crawling have a direct effect on how information is stored in the search index, which in turn affects how users can find certain topics and sources. The objective of this article is to provide an examination of the evolution of web crawling systems, with a particular focus on crawl scheduling and URL prioritization mechanisms.

Main part. The evolution of web crawling systems in search platforms

Contemporary search engines are seriously dependent on the quality and efficiency of their web crawling systems. These are automated components responsible for discovering, retrieving, and updating information from publicly accessible online sources. Web crawling refers to the systematic traversal of the internet to collect content for subsequent indexing. The architecture and performance of the tool directly affect the breadth of web coverage, freshness of crawled content and the relevance of search results. This dependency is rooted in the constantly expanding scale of the internet, since it continues to grow both in terms of the absolute number of websites and the relative complexity of their structure (fig. 1).

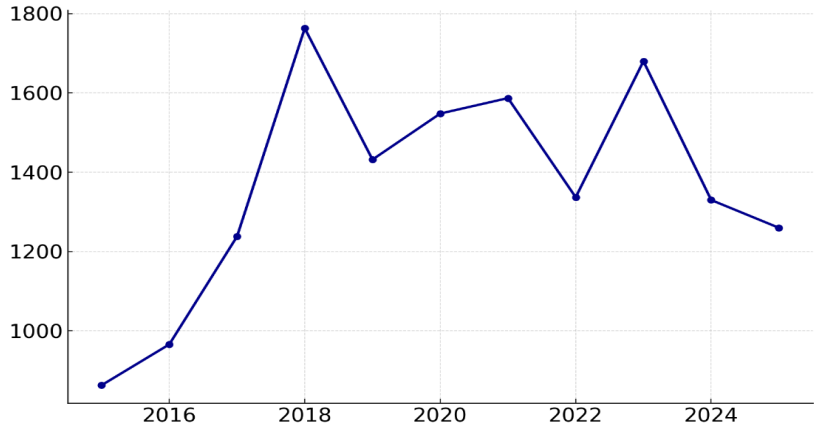


Figure 1. Estimated number of websites worldwide, millions [1]

At the beginning of the internet, web crawlers were simple programs that explored the web using basic graph search methods like depth-first and breadth-first search, with local state on a single machine [2]. As the internet grew and the number of websites increased, it became impossible to crawl all pages, so crawlers had to choose which URLs to crawl and process. They used host statistics and link statistics to achieve that.

Eventually, one crawler could not handle the task alone, even when crawling the best-ranked URLs. Crawlers were sharded to process non-overlapping subsets of the internet. This not only helped to store and process many more URLs but also made the process parallel. Each crawler cycle worked like this: process outgoing links from previously crawled sites, send links to the right shards, pick the best URLs to download, then download them. Sharding by host made counting host statistics cheaper. Later, very large websites appeared that were too big for one host. This led to a new stage where multiple shards handled one host, making host statistics harder to calculate.

Over time, different types of crawlers were made – sharded-by-host batch crawlers with a task scheduler like cron, real-time distributed systems with state storage on the same machine, and batch crawlers using MapReduce for data preparation, processing, and aggregation. But the best solution was using sharded event queues.

Modern web crawling systems are built on distributed, multi-component frameworks like message queues and NoSQL storage. They are capable of handling billions of URLs and terabytes of data daily. One major architectural advancement has been the adoption of asynchronous request handling, which significantly reduces delays caused by network operations. With asynchronous I/O,

crawlers can maintain tens of thousands of concurrent connections, optimizing system resource utilization [3]. Additionally, modern web crawling pipelines incorporate **multi-layered processing.** These architectural improvements also extend to scheduling mechanisms: a widely adopted design pattern is the use of hierarchical URL queues, which enables scalable and modular control over crawl execution.

An equally major component in the effectiveness of latest web crawling lies in the integration between the crawling subsystem and other elements of the search engine infrastructure. Data derived from user queries, click behavior or interaction patterns can inform and refine crawling strategies. This approach elevates the priority of URLs that are likely to hold greater value for users.

Web crawling faces a **number of problems** driven by both technical and structural changes in the web ecosystem. One of the most pressing issues is **scale**. Millions of new pages are added daily, but not all warrant inclusion in the index. This necessitates increasingly accurate estimation of content utility even before fetching. The increasing use of dynamically generated content, which makes pages fully accessible only after running JavaScript or interacting with client-side elements, adds to the complexity. Modern crawlers are capable of executing JavaScript on well-prepared pages, which allows them to retrieve accurate content. However, this is a resource-intensive operation and is therefore applied selectively to URLs where JavaScript rendering is essential.

In summary, the progression of web crawling systems represents advancements in architecture and data processing. It also means an ongoing adaptation to the changing digital environment. An effective crawler must be scalable, intelligent, and capable of adapting in defiance of high content volatility.

Crawl scheduling: load strategies and resource management

Once a distributed and scalable web crawling system is implemented, the process is faced with the growing but no less critical **challenge of effective crawl scheduling**. Since the amount of available web content can easily overwhelm even the most powerful of systems, it becomes necessary to create careful strategies regarding which webpages are worthy of visiting, what time repeated access should be made, and what efficient use of available infrastructure resources should be achieved. Instead of having a rigid visitation schedule, modern crawl scheduling follows ranking algorithms. For each URL, the system will first determine whether it meets predetermined inclusion criteria. If so, there is a relevance score determined according to various metadata like freshness, update frequency, and structure position. All these rankings guide the selection process, allowing the system to give preference to high-value content while still maintaining control over the resource consumption.

In practice, hierarchical URL queueing is the most commonly used strategy in modern web crawlers. Under this approach, scheduling is managed through separate queues – typically organized by domain, priority, or network segment – which enable flexible distribution of crawling tasks. Importantly, metadata such as the timestamp of the last visit, estimated update frequency, structural depth, and other attributes are not passed through queues along with URLs [4]. Instead, each URL is associated with a persistent record stored separately, allowing the crawler to maintain historical context and make informed decisions during subsequent scheduling iterations. Queues can be organized by domain, priority level, or network segment, enabling flexible allocation and redistribution of resources. Host-based partitioning of queues proves especially valuable in large systems because it increases cache locality, reduces database connections and calls between services, and allows for effective load balancing between machines. This architecture became the norm when sites grew beyond the capacity of single-node crawlers to deal with them, and it became necessary to assign certain queues or resources to high volume “host-monoliths” that generate a heavy amount of URLs.

The scheduling process in web crawlers involves more than just deciding on the next URL to crawl. It is more of a multifaceted problem that combines algorithmic techniques, infrastructural constraints, signals of user intent, and other considerations involved in accessing web content. This includes not only selecting which URLs to fetch next, but also deciding which outgoing links extracted from previously downloaded pages should be followed and incorporated into the crawl

frontier. Skillful scheduling secures that resources are fully utilized and it also helps to build a more accurate and complete portrayal of the web in the index used by browsers.

Link normalization plays a pivotal role in this process, removing duplicates and syntactic variations (e.g., query parameters like ?utm_source=) to prevent redundant fetching of the same resource under different addresses. In practice, seemingly insignificant query parameters can generate excessively long URLs – particularly on marketplaces and trading platforms, where multiple tracking or filtering parameters are often appended without affecting the actual content. Normalization helps eliminate such inflated variants, ensuring that crawlers do not repeatedly fetch semantically identical pages. Complementary to this, **content-level deduplication mechanisms** are employed to avoid indexing identical or near-identical pages [5].

Modern crawlers increasingly rely on **predictive models to anticipate content changes.** The goal is to determine the optimal moment for re-crawling a page. Visiting too frequently results in unnecessary resource expenditure, whereas infrequent revisits may lead to stale entries in the search index. Probabilistic models are constructed using historical change data, resource type, site structure, and external signals such as user activity or timestamp patterns.

An essential aspect of crawl scheduling involves **strict adherence to externally imposed constraints.** These limits can whether be defined by website administrators or arising from technical considerations. The standard robots.txt file specifies which URLs are permitted or not allowed for crawling. If a website is crawled too intensively, it may block the crawler's IP address or deny further access to its content. To prevent this, modern crawlers typically implement internal default rate limits for each site, which can be dynamically adjusted based on the site's responsiveness or trust level.

At the same time, the crawl-delay directive sets a minimum interval between successive requests to the same server. Although it is not officially specified in RFC 9309, this directive is nevertheless supported by many crawlers as a de facto standard.

Another important factor building crawl strategy is the **presence of active defense mechanisms** designed to reduce automated traffic. Crawlers that fail to comply with expected behavioral norms may be flagged as suspicious and subjected to blocking or CAPTCHA tests. In recent past, quite sophisticated traffic filtering systems have developed. One notable example is Cloudflare's **Block AI Bots** feature, a single-click solution marketed to non-technical users. While its ease of use is evident, the feature operates as a "black box", offering little transparency into its internal logic. As a result, even legitimate crawling agents may be denied access based on their technological signatures or user-agent identifiers (fig. 2).

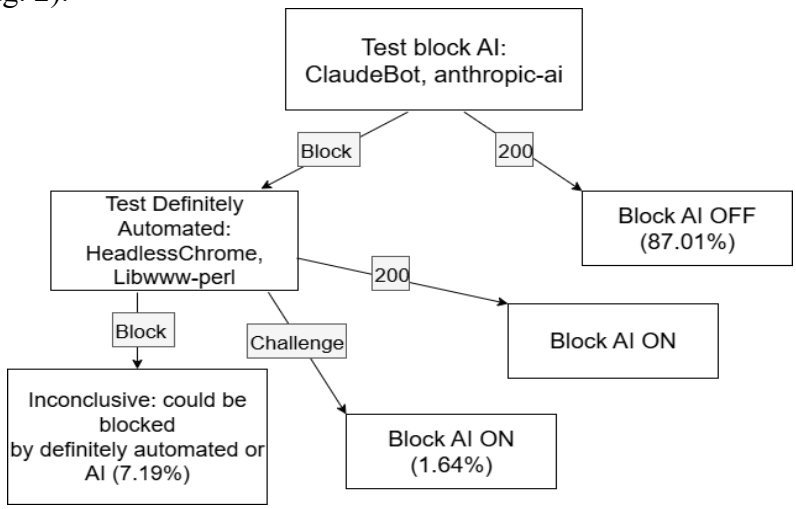


Figure 2. Diagram of decision outcomes for automated agents under the Cloudflare AI blocking mechanism [6]

These factors illustrate that crawl scheduling is no longer a question of simply optimizing internal resource usage – it also requires external sensitivity and responsiveness. Effective schedulers now must factor in infrastructural scaleability, dynamic content patterns, compliance with evolving access policies, and the presence of automatic defense mechanisms. As the web becomes even more

intricate, large-scale crawling sustainably will depend not only on better algorithms, but on the ability to navigate an increasingly regulated and adversarial terrain.

URL prioritization: impact on coverage and indexing efficiency

Since web crawling infrastructures suffer from inherent resource restrictions, it is necessary to develop a method for sequencing pages to crawl and prioritizing URLs. Unlike crawl scheduling, whose decision making is based basically on technical restrictions and queueing policies, the URL prioritization only concerns examining the utility and the relevance of web pages.

One of the foundational approaches to prioritization is based on **hyperlink popularity.** URLs that are linked to by a large number of external or authoritative sources are typically considered more significant. Such metrics are used in algorithms like PageRank and inform not only search result ranking but also the order in which URLs are crawled. The frequency of content updates is another important factor in URL prioritisation. Pages that change often, such as the front pages of news sites, update feeds, or product catalogues, should be crawled more often. To accomplish this, crawlers maintain search logs, logs of content changes, and aggregated host data, which are then used to predict the future update behavior of each resource.

In addition to both temporal and structural indicators, prioritization may also consider the topical **relevance of a page's content**. This has led to the spreading adoption of focused crawlers. These are specialized systems designed to gather information within a predefined domain of interest. A core component of focused crawling is the topic classifier, which evaluates the content of downloaded pages in real time to determine whether to continue following outbound links (fig. 3).

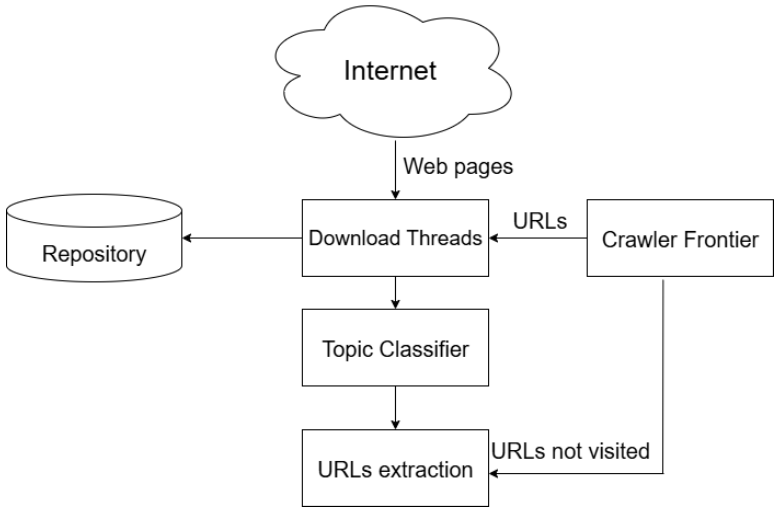


Figure 3. General flow of focused web crawler [7]

A website's navigational structure often reflects the relative importance of its pages within the domain, and this can serve as a basis for internal URL ranking. All of the aforementioned signals can be aggregated into a composite priority score, which guides decisions on the order of initial downloads as well as subsequent recrawling. The effectiveness of URL prioritization is not measured directly. Rather, it manifests through secondary metrics that characterize the overall state and behavior of the search system. These metrics can include index freshness or indexing latency (table 1).

Table 1

URL prioritization efficiency metrics [8, 9]

Metric	Description
Index freshness	The degree to which indexed data reflects the current state of websites. Effective prioritization helps timely update of frequently changing and important pages.
Coverage completeness	The proportion of valuable content discovered within the limits of crawl capacity. Poor prioritization can lead to skipping useful pages in favor of less important ones.

Metric	Description
Indexing latency	The delay between a page appearing on the web and being available in search. Smart prioritization reduces this delay for high-priority content.
Search result effectiveness	Although not directly dependent on crawling, search quality is strongly influenced by which pages are indexed and how often. Skipped or outdated pages reduce relevance and user satisfaction.

To evaluate prioritization effectiveness, practitioners rely on experimental measurements, A/B testing of competing strategies, and retrospective analysis of crawl logs and user interaction data.

Given the complexity of the modern web and the limitations of manually tuning prioritization rules, machine learning (ML) is increasingly integrated into crawling systems, as previously noted. These models are trained on historical crawl data, search query logs, and user behavior signals to predict the likely importance of web pages. Feature sets for such models may include the structure and length of the URL, semantic elements in the HTML (e.g., headers, meta tags), historical change patterns, and similar attributes [10].

The models estimate the probability that a URL is valuable and assign priority scores accordingly, which are then integrated into the global crawl queue. In some advanced implementations, reinforcement learning techniques can be applied, as they allow the system to adjust priorities dynamically based on downstream outcomes.

The integration of artificial intelligence can substantially increase the adaptability of crawling systems and enable them to better accommodate the heterogeneity of the web. Through incorporation of ML, crawlers can generalize beyond hand-crafted rules and adjust to emerging patterns in content and linking behavior.

Conclusion

The evolution of web crawling, as a core technical component of search engine operation, is representative of the overall effort to increase scalability and responsiveness in the rapidly growing environment of the internet. Older systems, based on linear navigation techniques, have been replaced by modern distributed and asynchronous systems, along with significant changes in the structure of web crawlers. These developments have made it possible to achieve dramatic improvements in both the breadth and pace of web crawler activities. However, with these advances in architecture, the significance of algorithmic decision-making, particularly crawl scheduling and resource allocation, has dramatically increased.

At the heart of modern-day web crawling techniques lies prioritization of URLs, a function that identifies the importance, value, and timeliness of web page indexing. Predictive models and various algorithms are increasingly being used to enhance indexing quality, incorporating behavioral signals and structural signals. At the same time, the increasing use of server-side protection mechanisms in the form of anti-bot systems and headless browser restrictions has added another layer of intricacy. Crawl planning and prioritization have thus evolved into tasks that are not only computationally demanding but also legally nuanced.

References

1. How Many Websites Are There in the World? / Siteefy // URL: https://siteefy.com/how-many-websites-are-there/ (date of application: 10.08.2025).

2. Kuznetsov I.A., Bobunov A.Yu., Bushuev S.A., Smirnov A.P., Pshichenko D.V. Integration of Big Data into Recommendation Systems: Content Personalization Technologies // Competitiveness in the Global World: Economics, Science, Technology. 2024. № 9. P. 56-61.

3. Garifullin R. Application of RxJS and NgRx for reactive programming in industrial web development: methods for managing asynchronous data streams and application state // International Journal of Professional Science. 2024. № 12-2. P. 42-47.

4. Mehyadin A.E., Abdulrahman L.M., Ahmed S.H., Qashi R. Distributed fundamentals based conducting the web crawling approaches and types (focused, incremental, distributed, parallel, hidden web, form focused and breadth first) crawlers // Journal of Smart Internet of Things. 2023. Vol. 2022(1). P. 10-32.

5. Viji D., Revathy S. Hash-Indexing Block-Based Deduplication Algorithm for Reducing Storage in the Cloud // Comput. Syst. Sci. Eng. 2023. Vol. 46(1). P. 27-42.

6. Liu E., Luo E., Shan S., Voelker G.M., Zhao B.Y., Savage S. Somesite I Used to Crawl: Awareness, Agency and Efficacy in Protecting Content Creators from AI Crawlers. In Proceedings of the 2025 ACM Internet Measurement Conference (IMC '25). Madison, WI, USA. ACM, New York, NY, USA, 22 p.

7. Neelakandan S., Arun A., Bhukya R.R., Hardas B.M., Kumar T.C., Ashok M. An automated word embedding with parameter tuned model for web crawling // Intelligent Automation & Soft Computing. 2022. Vol. 32(3). P. 1617-32.

8. Sethi S. An optimized crawling technique for maintaining fresh repositories // Multimedia Tools and Applications. 2021. Vol. 80(7). P. 11049-77.

9. Rafsanjani A.S., Kamaruddin N.B., Behjati M., Aslam S., Sarfaraz A., Amphawan A. Enhancing malicious URL detection: A novel framework leveraging priority coefficient and feature evaluation // IEEE Access. 2024. Vol. 12. P. 85001-26.

10. Kunekar P., Nimbolkar A., Patil A., Lakde V., Wadile P., Rathod K. Product Review Sentiment Analysis using Web Crawler and Machine Learning. // Grenze International Journal of Engineering & Technology (GIJET). 2024. Vol.10(2). № 5479.