Программные системы и вычислительные методы
*Правильная ссылка на статью:*
Ратушняк Е.А. Анализ современных методов оптимизации в React // Программные системы и вычислительные методы. 2025. № 3. DOI: 10.7256/2454-0714.2025.3.75278 EDN: CFXZBG URL: https://nbpublish.com/library_read_article.php?id=75278

# Анализ современных методов оптимизации в React

**Ратушняк Евгений Алексеевич**

ORCID: 0009-0006-3609-4194

студент; факультет программной инженерии и компьютерной техники; Национальный исследовательский университет ИТМО

197101, Россия, г. Санкт-Петербург, Петроградский р-н, Кронверкский пр-кт, д. 49

evgrat123@mail.ru

Статья из рубрики "Математическое и программное обеспечение новых информационных технологий"

**DOI:**

10.7256/2454-0714.2025.3.75278

**EDN:**

CFXZBG

**Дата направления статьи в редакцию:**

20-07-2025

**Дата публикации:**

27-07-2025

**Аннотация:** В статье проведён комплексный анализ методов оптимизации производительности React приложений: мемоизации (React.memo, useMemo, useCallback), виртуализации списков (react window, react virtualized), разделения кода (React.lazy, Suspense), оптимизации Context API и новых возможностей React18 (автоматический батчинг, Concurrent Mode). Приведённый анализ и данные о производительности основываются на различные научные исследования, а также на документацию самих методов оптимизации. Для измерения производительности React приложений разработчикам рекомендуется использовать React Profiler API и Chrome Performance. Разработано тестовое SPA с динамической фильтрацией данных и трёх вариантами рендеринга. С помощью ReactProfiler API измерено время обработки от 1000 до 20000 элементов десять сходящихся раз с последующей статистической обработкой. Методология включает комплексный теоретическое анализ, разбор механизмов

различных методов оптимизаций обновлений и их влияние на производительность. Научная новизна статьи заключается в комплексном анализе и практическом сравнении ключевых подходов к оптимизации во фреймворке React. Практическая значимость работы обосновывается тем, что результаты могут быть использованы непосредственно в коммерческой разработке программного обеспечения. Также в статье было проведено экспериментальное сравнение библиотек виртуализации списков с использованием компьютерного эксперимента, с последующим статистическим анализом. Результаты показали: react window обеспечивает до 95% прироста скорости и стабильность при росте нагрузки, а react virtualized предлагает расширенный функционал ценой слегка большей латентности, что подтверждается другими исследованиями. Статья содержит не только теоретическое описание, но и практические примеры, раскрывающие способы оптимизации в реальных приложениях, что подтверждает практическую значимость.

**Ключевые слова:**

оптимизация, React, производительность, одностраничные приложения, ленивая загрузка, разделение кода, react-window, react-virtualized, профилирование, веб-интерфейсы

## Введение

В условиях стремительного роста сложности веб-приложений и требований к их производительности особую актуальность приобретает задача оптимизации пользовательских интерфейсов. Современные одностраничные приложения (SPA [1]), построенные с использованием библиотеки React [2, 11, 12], зачастую оперируют большими объёмами данных и сложной компонентной иерархией, что может приводить к значительным издержкам на повторные рендеры, задержкам интерфейса и ухудшению пользовательского опыта. В связи с этим разработка и внедрение эффективных стратегий оптимизации рендеринга становится важным направлением как в индустрии, так и в научно-прикладных исследованиях в области frontend-разработки.

Научная новизна настоящего исследования заключается в комплексном анализе и практическом сравнении ключевых подходов к оптимизации производительности в React, включая мемоизацию компонентов, виртуализацию списков, разделение кода, оптимизацию Context API, а также возможности, предоставляемые React 18, такие как автоматический батчинг и Concurrent Mode. Особое внимание уделяется сравнительной оценке двух широко используемых библиотек виртуализации — react-window и react-virtualized — в контексте реального применения на примере веб-приложения с динамической фильтрацией данных.

Практическая значимость работы состоит в том, что полученные результаты могут быть непосредственно использованы при разработке высокопроизводительных веб-интерфейсов, особенно в проектах с большим количеством отображаемых элементов, таких как таблицы, ленты новостей и административные панели. Представленные рекомендации и эмпирические данные позволяют сделать обоснованный выбор технологий и инструментов для повышения отзывчивости и масштабируемости интерфейсов.

Цель исследования — провести комплексный анализ и экспериментальную оценку эффективности различных методов оптимизации рендеринга в React-приложениях.

Объектом исследования является клиентские фреймворки, а предметом исследования выступают оптимизации производительности в React.

**Мемоизация компонентов в React**

React.memo – это HOC-компонент, обёртывающий функциональный компонент. Он кэширует результат рендеринга, если входные пропсы не изменились, что позволяет уменьшить количество повторных рендеров. Сравнение пропсов при этом выполняется по ссылке. На практике применение React.memo значительно сокращает число лишних рендеров [13]: согласно эмпирическому исследованию, React.memo может снизить частоту повторных рендеров на 40–60%[3]. Это особенно полезно для чистых дочерних компонентов с неизменяемыми пропсами или статические компоненты, которые без мемоизации иначе перерисовывались бы при каждом обновлении родительского компонента.

Хуки useMemo и useCallback предназначены для локальной оптимизации[4]. Хук useMemo кэширует результат дорогостоящей функции и возвращает закэшированное значение, если зависимости не изменились. Это позволяет экономить время вычислений: по оценкам, использование useMemo может снизить затраты CPU на 25–35% при тяжёлых вычислениях. Хук useCallback сохраняет ссылку на функцию между рендерами [14], предотвращая её пересоздание. Он особенно эффективен, когда функция передаётся во вложенные компоненты или используется в зависимостях других хуков. Такой подход позволяет улучшить производительность до 25%.

Хотя мемоизация повышает производительность, её использование имеет издержки. Во-первых, дополнительный кэш требует памяти (примерно 0.2–0.4 КБ на компонент для React.memo) и CPU на сравнение зависимостей. Во-вторых, чрезмерное применение React.memo/useMemo/useCallback усложняет код и усложняет его поддержку. Поэтому хуки useMemo и useCallback стоит применять избирательно — только там, где выявлено реальное узкое место по производительности.

**Библиотеки виртуализации в React**

В экосистеме React для виртуализации списков наиболее распространены библиотеки react-window и react-virtualized. Обе предлагают готовые компоненты для отображения длинных списков без перегрузки DOM. Так, например, библиотека react-window ориентирована на простые сценарии с фиксированным размером элементов. Она проще и компактнее, но требует, чтобы высота/ширина ячеек была неизменной. В свою очередь react-virtualized предоставляет более богатый функционал: динамический подсчёт размеров ячеек, ленивую загрузку данных при скролле, синхронизацию прокрутки между несколькими списками.

· react-window: облегчённая библиотека для windowing. Быстрая, с простым API, но предполагает фиксированные размеры элементов и плоскую структуру списка.

· react-virtualized: более тяжёлая и функциональная. Поддерживает переменную высоту элементов, вложенные сетки и сложные сценарии (динамические размеры, синхронизация скролла, загрузку по мере прокрутки).

**Разделение кода**

Code splitting (разделение кода) используется для минимизации времени загрузки приложения путём отделения второстепенных модулей от основного бандла [5]. В React

данная функциональность реализована через механизм динамического импорта, а именно — с помощью React.lazy() и компонента Suspense. Такой подход позволяет загружать компоненты по требованию, сокращая размер initial bundle до 50% [6] и ускоряя начальное отображение страницы.

При разделении кода даже на небольшие чанки возможно уменьшение времени первого рендера (First Contentful Paint) на 15–30%. Применение Suspense обеспечивает мягкую деградацию пользовательского опыта, предоставляя визуальные индикаторы ожидания загрузки. Тем не менее, данный механизм имеет ограничения: не поддерживает серверный рендеринг по умолчанию, требует жёсткого контроля точек разделения и может усложнить архитектуру при обилии вложенных асинхронных компонентов.

**Оптимизация Context API**

Context API позволяет реализовать глобальное состояние без использования сторонних библиотек, но он обладает существенным недостатком: любые изменения значения value провайдера вызывают повторный рендер всех потребителей вне зависимости от того, какие именно данные были изменены. В ряде исследований отмечается, что такая стратегия ведёт к существенным накладным расходам в интерфейсах с глубокой иерархией или частыми обновлениями состояний.

Одним из эффективных решений является декомпозиция глобального контекста на более узкоспециализированные подмодули (например, отдельные контексты для пользователя, темы, настроек и т.д.), что позволяет ограничить перерисовку только потребителями соответствующего значения [7]. Также предлагается комбинировать Context с селекторами или использовать внешние библиотеки, обеспечивающие fine-grained подписки на изменения состояния.

**Батчинг и Concurrent Mode**

С выходом React 18 были введены фундаментальные изменения в стратегию управления рендерами. Одной из важнейших новаций стал автоматический батчинг, расширивший область действия объединения setState-вызовов за пределы синтетических событий — теперь в единый рендер объединяются также обновления из setTimeout [15], fetch [16], и других асинхронных контекстов. Это позволяет снизить количество рендеров и увеличить плавность интерфейса.

Вторая ключевая особенность React 18 — это Concurrent Mode, включающий механизм time slicing и асинхронного рендеринга. Он позволяет приостанавливать тяжелые вычисления в интерфейсе для обработки приоритетных задач, таких как ввод с клавиатуры или клики. Механизм реализован с помощью функций startTransition, useTransition, useDeferredValue. Эти средства позволяют разработчику вручную обозначить «некритичные» обновления, которые могут быть отложены без ущерба для UX. Исследования показывают, что внедрение Concurrent Mode снижает показатель Total Blocking Time до 40% в многофункциональных SPA.

**Инструменты профилирования и метрики**

Чтобы убедиться в эффективности оптимизаций, применяют профилирование и бенчмаркинг. React предоставляет Profiler [8, 9], который вызывает функцию обратного вызова при каждом коммите и передаёт времена рендеринга: текущую и базовую. Сравнивая их, можно понять, насколько снизилась нагрузка за счёт мемоизации и прочих оптимизаций.

Кроме того, используют встроенные инструменты браузера. В Chrome DevTools есть Performance-панель, где меряют FPS, время компоновки и т.п. Для веб-страниц часто отслеживают Core Web Vitals: First Contentful Paint, Time to Interactive, First Input Delay [10, 17].

Также полезны нагрузочное тестирование и эмуляция разных сетей. На практике хорошо фиксировать метрики до и после оптимизаций. Как правило, продакшн-сборки React [18] работают быстрее: профилирование показывает, что продакшн-сборки показывают производительность на 25–35% лучше, чем сборки в режиме разработчика.

Сравнение библиотек react-virtualized и react-window В исследовании проводится сравнительный анализ эффективности оптимизаций, реализованных в библиотеках виртуализации интерфейсов — react-virtualized и react-window. С этой целью было разработано веб-приложение, предназначенное для табличного отображения данных с поддержкой динамической фильтрации с использованием строки поискового запроса. Такой подход позволяет эмпирически оценить производительность и ресурсоэффективность каждой из библиотек при работе с большими объёмами данных. На рисунке 1 представлен код, который реализует мемоизированную фильтрацию и сортировку массива объектов на основе заданного поискового запроса и направления сортировки. Сначала происходит фильтрация элементов массива items путём поиска подстроки запроса в заголовке каждого элемента без учёта регистра. Основываясь на параметре сортировки, результирующий массив инвертируется для обеспечения убывающего порядка. Использование хука useMemo гарантирует, что вычисления будут производиться только при изменении одной из зависимостей: запроса, сортировка или флаг сортировки, что способствует оптимизации производительности компонента.

```
const filteredItems = useMemo(() => {
  let filtered = items.filter((item) => item.title.toLowerCase().includes(query.toLowerCase()));
  if (!sortAsc) {
    filtered = [...filtered].reverse();
  }
  return filtered;
}, [query, sortAsc, items]);
```

Рисунок 1 – Фильтрация элементов с использованием мемоизации

На рисунке 2 представлен код, который реализует асинхронное обновление состояния с использованием хука useTransition [19]. При изменении значения в поле ввода вызывается функция обработки нового значения, которая инициирует переходное обновление состояния запроса через функцию startTransition. Это позволяет отнести данное обновление к задачам с низким приоритетом, снижая нагрузку на главный поток и сохраняя отзывчивость пользовательского интерфейса во время интенсивных операций. Переменная isPending отражает текущее состояние перехода, что позволяет при необходимости отображать индикатор загрузки.

```
const [isPending, startTransition] = useTransition();

const handleChange = (e) => {
  const value = e.target.value;
  startTransition(() => {
    setQuery(value);
  });
};
```

Рисунок 2 – Пример использования useTransition Пример работы разработанного сайта на рисунке 3.

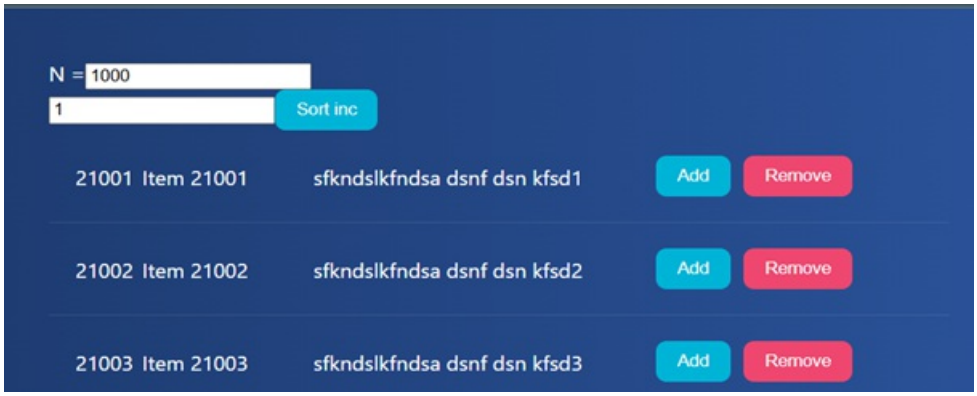


Рисунок 3 – Пример работы приложения

В базовой версии разрабатываемого приложения осуществляется рендеринг статического массива, содержащего N элементов. Также реализован механизм фильтрации, при котором ввод запроса в текстовое поле инициирует динамическую перерисовку значительного числа элементов пользовательского интерфейса в режиме реального времени. Во втором и третьем вариантах приложения к тому же набору данных применяется технология виртуализации с использованием библиотек react-window и react-virtualized соответственно. Для всех трёх реализаций производится измерение времени рендеринга, что позволяет количественно оценить влияние виртуализации на производительность интерфейса при обработке больших объёмов данных.

Каждая из реализованных версий приложения проходит не менее десяти независимых запусков с целью обеспечения статистической надёжности результатов. В качестве инструмента для измерения производительности используется React Profiler API [20], который позволяет осуществлять точечные замеры времени рендеринга отдельных компонентов, обеспечивая детализированную оценку эффективности визуализации в различных сценариях.

Результаты скорости применения фильтра к данным представлены в таблице 1.

Таблица 1 – Результаты измерения производительности виртуализации

| Количество элементов, шт | Базовая версия, мс | Виртуализация списка react-window, мс | Виртуализация списка react-virtualized, мс |
|---|---|---|---|
| 1000 | 94 | 5 | 9 |
| 2000 | 153 | 4 | 10 |
| 3000 | 174 | 4 | 10 |
| 4000 | 221 | 4 | 10 |
| 5000 | 264 | 4 | 10 |
| 7500 | 289 | 6 | 9 |
| 10000 | 343 | 5 | 10 |
| 12500 | 428 | 6 | 11 |
| 15000 | 545 | 6 | 11 |
| 17500 | 669 | 7 | 11 |
| 20000 | 782 | 7 | 10 |

Анализ экспериментальных данных позволяет сделать вывод о более высокой производительности библиотеки react-window по сравнению с react-virtualized. При этом

эффективность react-window остаётся стабильной вне зависимости от общего количества элементов в списке. Дополнительно, применение механизмов виртуализации прокрутки обеспечивает отображение исключительно тех элементов, которые находятся в пределах видимой области (viewport), что существенно снижает нагрузку на систему и обеспечивает значительное повышение производительности по сравнению с базовой реализацией.

**Заключение**

В данной работе был проведён обзор и эмпирический анализ современных подходов к оптимизации производительности пользовательских интерфейсов на базе React. Рассмотренные технологии — мемоизация, виртуализация списков, разделение кода, оптимизация Context API, а также возможности React 18, а именно автоматический батчинг и Concurrent Mode — играют ключевую роль в построении отзывчивых и масштабируемых веб-приложений.

В экспериментальной части особое внимание было уделено сравнению производительности двух популярных библиотек виртуализации: react-window и react-virtualized. Результаты тестирования, выполненные с использованием React Profiler, показали, что react-window обеспечивает наилучшую производительность при рендеринге больших списков — с минимальным временем отклика, сохраняющим стабильность даже при увеличении количества элементов до 20000. В то время как react-virtualized демонстрирует более высокую универсальность, его производительность в ряде случаев незначительно уступает react-window.

## Библиография

1. Iseal, S. Edge Computing and React: Enhancing Performance at the Edge. – 2025. – С. 24.
2. Сугаипов, С. А. А., Гериханов, З. А. Оптимизация веб-разработки с помощью React, Angular и Vue // Тенденции развития науки и образования. – 2023. – № 98-10. – С. 141-143. – DOI 10.18411/trnio-06-2023-565. – EDN HYNKWA.
3. Veeranjaneyulu, V. Performance Optimization Techniques in React Applications: A Comprehensive Analysis // International Journal of Research in Computer Applications and Information Technology. – 2024. – Т. 7, № 2. – С. 1165–1177.
4. Савкин, С. С., Логвинов, Д. В. Оптимизационные возможности JavaScript-библиотеки REACT 18 // Вызовы глобализации и развитие цифрового общества в условиях новой реальности: сборник материалов IV Международной научно-практической конференции, Москва, 19 декабря 2022 года. – Москва: Алеф, 2022. – С. 126-129. – DOI 10.34755/IROK.2022.97.75.019. – EDN ALNIUP.
5. Чэнь, К. Improving Front-end Performance through Modular Rendering and Adaptive Hydration (MRAH) in React Applications. – 2025. – Режим доступа: https://arxiv.org/abs/2504.03884, свободный.
6. Toprak, Ahmet, Toprak, Feyzanur. Improving and Optimizing React Web Applications: Strategies and Techniques. – 2025. – С. 17.
7. Пирюшов, А. С., Пирюшов, М. С. Сравнительный анализ времени рендеринга компонентов при использовании различных инструментов управления состояниями в React-приложениях // Международный журнал информационных технологий и энергоэффективности. – 2024. – Т. 9, № 3(41). – С. 79-87. – EDN KPOFLF.
8. Ким, А. Оптимизация ре-рендеринга компонентов инструментами React // Научно-технические инновации и веб-технологии. – 2022. – № 1. – С. 4-10. – EDN JHYSXA.
9. Dudak, A. Optimization of loading and performance in SPA on React // Тенденции

развития науки и образования. – 2024. – № 116-19. – С. 183-187. – DOI 10.18411/trnio-12-2024-899. – EDN JJQMMK.
10. Iseal, S. Performance Benchmarking Techniques for React Applications. – 2025. – С. 14.
11. Кравцов, Е. П. Разработка высокопроизводительных React-приложений: методы и практики оптимизации // European science. – 2024. – № 1 (69).
12. Яровая, Е. В. Нестандартные архитектура в написание веб приложений // Столыпинский вестник. – 2022.
13. Dubaj, S., Pańczyk, B. Comparative of React and Svelte programming frameworks for creating SPA web applications // Journal of Computer Sciences Institute. – 2022. – Т. 25. – С. 345-349. – DOI 10.35784/jcsi.3020. – EDN AYUSVM.
14. Karić, A., Durmić, N. Comparison of JavaScript Frontend Frameworks-Angular, React, and Vue // International Journal of Innovative Science and Research Technology (IJISRT). – 2024. – С. 1383–1390. – DOI 10.38124/ijisrt/ijisrt.
15. Kasenda, R., Tenda, J., Iman, E., Manantung, J., Moekari, Z., Pantas, M. The Role and Evolution of Frontend Developers in the Software Development Industry // Jurnal Syntax Admiration. – 2024. – Т. 5. – С. 5191–5196. – DOI 10.46799/jsa.v5i11.1852.
16. Karka, Narender. Front-End Performance Optimization: A Comprehensive Guide // International Journal of Scientific Research in Computer Science, Engineering and Information Technology. – 2025. – Т. 11. – С. 83-100. – DOI 10.32628/CSEIT251112389.
17. Mathew, Prakash. Front-End Performance Optimization for Next-Generation Digital Services // Journal of Computer Science and Technology Studies. – 2025. – Т. 7. – С. 993-1000. – DOI 10.32996/jcsts.2025.7.4.111.
18. Yan, Fenglong, Xu, Zhao, Zhong, Yu, HaiBei, Zhang, Ge, Chang. Research on Performance Optimization Scheme for Web Front-End and Its Practice. – 2021. – DOI 10.1007/978-981-15-8411-4_118.
19. Saks, E. JavaScript frameworks: Angular vs React vs Vue. – 2019.
20. Paakkanen, J. Upcoming JavaScript Web Frameworks and Their Techniques. – LUT University, 2022. – 62 с.

## Результаты процедуры рецензирования статьи

*В связи с политикой двойного слепого рецензирования личность рецензента не раскрывается.*
*Со списком рецензентов издательства можно ознакомиться здесь.*

Представленная статья на тему «Анализ современных методов оптимизации в React» соответствует тематике журнала «Программные системы и вычислительные методы» и посвящена вопросу оптимизации пользовательских интерфейсов. Данная тема приобретает особую актуальность в условиях стремительного роста сложности веб-приложений и требований к их производительности.
В статье представлен достаточно широкий анализ литературных российских и зарубежных источников по теме исследования.
Авторами в статье указана цель исследования, которая заключается в проведении комплексного анализа и экспериментальной оценки эффективности различных методов оптимизации рендеринга в React-приложениях. В качестве объекта исследования авторы указывают клиентские фреймворки, а предметом исследования выступают оптимизация производительности в React.
В качестве научной новизны авторами указан комплексный анализ и практическое сравнение ключевых подходов к оптимизации производительности в React, включая мемоизацию компонентов, виртуализацию списков, разделение кода, оптимизацию Context API, а также возможности, предоставляемые React 18, такие как автоматический

батчинг и Concurrent Mode.

Стиль и язык изложения материала является научным и доступным для широкого круга читателей. Статья по объему соответствует рекомендуемому объему от 12 000 знаков.

Статья достаточно структурирована - в наличии введение, заключение, внутреннее членение основной части (Мемоизация компонентов в React, Библиотеки виртуализации в React, Разделение кода, Оптимизация Context API, Батчинг и Concurrent Mode, Инструменты профилирования и метрики).

Теоретико-методологической основой исследования послужили работы таких авторов как Сугаипов, С. А. А., Гериханов, З. А.; Veeranjaneyulu, V. ; Савкин, С. С., Логвинов, Д. В; Пирюшов, А. С., Пирюшов, М. С.; Kasenda, R., Tenda, J., Iman, E., Manantung, J., Moekari, Z., Pantas, M. и др.

Практическая значимость работы как указывают авторы состоит в том, что полученные результаты могут быть непосредственно использованы при разработке высокопроизводительных веб-интерфейсов, особенно в проектах с большим количеством отображаемых элементов, таких как таблицы, ленты новостей и административные панели. Авторами представлены рекомендации и эмпирические данные, которые позволяют сделать обоснованный выбор технологий и инструментов для повышения отзывчивости и масштабируемости интерфейсов.

В заключении авторы указали результаты сравнения производительности двух популярных библиотек виртуализации: react-window и react-virtualized. Результаты тестирования, выполненные с использованием React Profiler, показали, что react-window обеспечивает наилучшую производительность при рендеринге больших списков — с минимальным временем отклика, сохраняющим стабильность даже при увеличении количества элементов до 20000. В то время как react-virtualized демонстрирует более высокую универсальность, его производительность в ряде случаев незначительно уступает react-window.

Статья «Анализ современных методов оптимизации в React» может быть рекомендована к публикации в журнале «Программные системы и вычислительные методы».