International Journal of Latest Engineering and Management Research (IJLEMR)
ISSN: 2455-4847
www.ijlemr.com || Volume 09 – Issue 05 || May 2024 || PP. 51-57

Performance Analysis of React Applications: Optimization Methods and Tools

Chakvetadze Vissarion Ruslanovich
Senior Software Engineer, Givelify
Tskaltubo, Georgia

Abstract: The article discusses the performance analysis of React applications. A research problem has been identified, which is that updating a tree is undesirable since updating a property at the top of the tree causes an update to all child components and elements. It is noted that with effective organization of system management, the development process can be significantly simplified and React application scalability can be ensured, which allows for effective management and reduction of the load on server resources. A review of the literature has been carried out that examines the analysis of React applications, various approaches that can be used in state management in web applications based on React, as well as software tools that make it possible to modernize the development process of web applications and websites through the use of third-party development tools in work, in particular, consideration was given to frameworks that represent the basis of many large web applications. Literature analysis showed that enough attention has been paid to performance analysis of React applications, while insufficient attention has been paid to optimization tools and methods. An overview of methods and tools that are designed to optimize a React application is presented. In this study, the following application optimization methods were considered: optimization of event handlers; using React. memo; using the React.Profiler API; virtualization of long lists; lazy loading of components; optimizing state updates using Immer; as well as the use of throttling and debouncing for input handlers. The conclusions drawn from the study will allow developers to quickly develop React applications without experiencing problems with their performance.

Keywords: application, optimization, performance, methods, tools, React, libraries, projects

Introduction

Even though React is a productive framework that makes it possible to quickly develop dynamic pages with a large number of elements, there are situations when there are so many such elements on the page that the built-in performance is not enough, in which case it becomes necessary to optimize it.

React is a popular JavaScript framework for creating user interfaces that use the concept of a virtual DOM to show state changes in subtrees of nodes. One of the performance problems in React applications is unwanted tree updates, since updating a property at the top of the tree forces an update to all child components and elements.

When developing React applications, its important to pay extra attention to how the data is stored and processed so it will be easier to adapt to changing requirements in the future. With effective organization of systems management significantly simplifies the development process and ensures the scalability of a React application, enabling effective management and reduction of server resource load.

Literature Review

T. M. Tielens [1] paid special attention to studying the analysis of React applications. In his research he describes the key points of working with the React framework, as well as the analysis of the life cycles of React applications and their architecture.

The works of K.A. Melikhova, E.S. Tomilin, and A.S. Kropotov [2] discusses different approaches that can be used in state management in web applications based on React. They review different technologies and management tools that are focused on the efficiency of the development process and ensuring the best application performance.

D.N. Vankevich in work [3] examined software tools that enable modernization of the process of developing web applications and websites through the use of popular development tools, with a focus on frameworks that form the foundation in many large web apps.

Literature analysis led to the conclusion that sufficient attention has been paid to performance analysis of React applications, while, on the contrary, insufficient attention has been paid to optimization tools and methods. In this connection, we can conclude about the importance of research in this direction.

Methodology

React applications by default guarantee a very fast user experience, however, as the application grows, developers may encounter some performance issues that will slow down the applications. Such problems usually result from the wrong approach to application development.

To improve the performance of React applications, developers can use special optimization methods and tools. This study will consider the following optimization methods: optimization of event handlers; using React.memo; Using the React.Profiler API; virtualization of long lists; lazy loading of components; optimizing state updates using Immer; as well as the use of throttling and debouncing for input handlers. The tools will be reviewed by Gatsby, Webpack, Preact, Next.js, Storybook, and Razzle.

Results

1. Optimization methods

Performance optimization is a critical aspect affecting web application development. In the context of React applications, several optimization techniques can be used to significantly improve the user experience. This is primarily achieved by reducing loading times and improving response efficiency. Below we'll look at different optimization techniques, providing a detailed overview of each technique, its necessity, and the specific scenarios in which it is most useful:

1. Event Handler Optimization (Figure 1) is a technique designed to improve performance by preventing unnecessary re-rendering of child components that depend on memoized event handlers. This technique is very important for scenarios in which complex interactive elements frequently trigger event handlers. By optimizing event handlers, developers can ensure that only necessary components are redrawn in response to user interaction, minimizing performance overhead and improving the experience.

```
import { useState, useCallback } from 'react';

const Counter: React.FC = () => {
    const [count, setCount] = useState(0);
    const increment = useCallback(() => {
        setCount((prevCount) => prevCount + 1);
    }, []);

    return (
        <div>
            <p>Count: {count}</p>
            <button onClick={increment}>Increment</button>
        </div>
    );
};
```

Figure 1 – Example of using the event handler optimization method

2. Virtualization of long lists (Fig. 2). Using the react-window library to virtualize long lists is an efficient method of displaying only visible elements to the user while reducing the number of DOM elements created and updated. This approach is especially useful in scenarios where applications display extensive lists of data, such as product inventory or search results. Virtualization improves application performance by limiting the amount of DOM manipulation required, thereby speeding up rendering times and reducing resource consumption.

```tsx
import { FixedSizeList as List } from 'react-window';

const Row: React.FC<{ index: number; style: React.CSSProperties }> = ({
    index,
    style,
}) => {
    return (
        <div style={style}>
            <p>{`Row ${index}`}</p>
        </div>
    );
};

const VirtualizedList: React.FC = () => {
    const itemCount = 1000;

    return (
        <List height={500} itemCount={itemCount} itemSize={50} width={300}>
            {Row}
        </List>
    );
};
```

Figure 2 – Example of using the long list virtualization method

3. React.memo to prevent unnecessary renders (Fig. 3). React.memo is a Higher Order Component (HOC) that prevents unnecessary re-renders of functional components. It renders only when props change, making it invaluable for complex components that require significant rendering resources. Using React.memo is recommended in situations where components receive frequently changing props but don't always need to update their representation, saving on rendering costs and improving overall performance.

```tsx
import { memo } from 'react';

interface Props {
    title: string;
    subtitle: string;
}

const MyComponent: React.FC<Props> = memo(({ title, subtitle }) => {
    return (
        <div>
            <h1>{title}</h1>
            <h2>{subtitle}</h2>
        </div>
    );
});
```

Figure 3 – Example of using React.memo

4. Measuring performance using React.Profiler (Fig. 4). The React.Profiler API makes it easy to measure component performance by collecting timing information at each rendering stage. Using React.Profiler is necessary to identify and eliminate performance bottlenecks in React applications. This method allows developers to gain insight into the rendering behavior of their components, enabling targeted optimizations to improve efficiency and responsiveness.

```
import { Profiler } from 'react';

const App: React.FC = () => {
    const onRender = (
        id: string,
        phase: 'mount' | 'update',
        actualDuration: number,
        baseDuration: number,
        startTime: number,
        commitTime: number,
        interactions: Set<{ id: number; name: string; timestamp: number }>,
    ) => {
        console.log('Profiler:', {
            id,
            phase,
            actualDuration,
            baseDuration,
            startTime,
            commitTime,
            interactions,
        });
    };

    return <Profiler id="MyComponent" onRender={onRender}></Profiler>;
};
```

Figure 4 – Example of using the React.Profiler API

5. Throttling and debouncing for input handlers (Fig. 5). Implementation of throttling and debouncing methods for input handlers effectively limits the speed of function execution. Throttling directly reduces the frequency at which functions fire, and debugging ensures that a function is executed only once after a period of inactivity. These techniques are very important for optimizing performance in scenarios involving user input, such as search strings or scroll events, by reducing the frequency of function calls and thus minimizing the performance impact.

```
import { useState, useCallback } from 'react';
import { debounce } from 'lodash-es';

const SearchBox: React.FC = () => {
    const [searchTerm, setSearchTerm] = useState('');

    const handleSearch = useCallback(
        debounce((value: string) => {
            console.log('search', value);
        }, 300),
        [],
    );

    const handleChange = (event: React.ChangeEvent<HTMLInputElement>) => {
        const value = event.target.value;

        setSearchTerm(value);
        handleSearch(value);
    };

    return (
        <div>
            <input onChange={handleChange} type="text" value={searchTerm} />
        </div>
    );
};
```

Figure 5 – Example of using throttling and debouncing for input handlers

6. Lazy loading of components (Fig. 6). Lazy loading is a strategy that delays loading non-critical components until they are needed. This technique, implemented with React.lazy and Suspense, significantly improves the initial load time of applications by reducing the amount of JavaScript that needs to be loaded and parsed. Lazy loading is especially useful in applications that include heavy components that are not necessary for the initial user experience, providing a faster and more efficient loading process.

```
import { lazy, Suspense } from 'react';
const LazyLoadedComponent = lazy(() => import('./LazyLoadedComponent'));

const App: React.FC = () => {
    return (
        <div>
            <Suspense fallback={<div>Loading...</div>}>
                <LazyLoadedComponent />
            </Suspense>
        </div>
    );
};
```

Figure 6 - Example of using the lazy loading method of components

7. Optimization of state updating using Immer (Fig. 7). Immer is a popular library that makes working with immutable data structures in JavaScript easier by providing a method for optimizing state updating. By making it easier to work with immutable structures, Immer can be useful in scenarios that require frequent and complex state updates. This approach not only improves the readability and maintainability of the code but also

improves performance by ensuring that only the necessary data is updated, reducing the overhead associated with changing state.

```tsx
import { useState } from 'react';
import { produce } from 'immer';

interface User {
    id: number;
    name: string;
}

const App: React.FC = () => {
    const [users, setUsers] = useState<User[]>([
        { id: 1, name: 'Alice' },
        { id: 2, name: 'Bob' },
    ]);

    const updateUser = (id: number, newName: string) => {
        setUsers(
            produce((draftUsers: User[]) => {
                const user = draftUsers.find((el) => el.id === id);
                if (user) {
                    user.name = newName;
                }
            }),
        );
    };
```

Figure 7 – Example of using the state update method using Immer

Each of these optimization methods serves a specific purpose and is suitable for certain React application development scenarios (Table 1). By understanding and applying these techniques correctly, developers can significantly improve the performance and user experience of their web applications.

Table 1. Summary table of optimization methods

Instrument	**Brief Description**	**Suitable For**
Event Handlers	Optimizes event handlers to prevent unnecessary re-renders	Scenarios with frequent and complex user interactions, where it is necessary to avoid extra re-renders
react-window	Displays only the visible elements of lists, reducing the number of DOM elements	Applications displaying long lists of data, where it's important to reduce the load on the DOM and improve performance
React.memo	Prevents unnecessary re-renders of components, re-renders only when props change	Complex components with frequently changing props, requiring performance optimization
React.Profiler	Measures the performance of components, helping to identify bottlenecks	Analyzing and optimizing the performance of components in React applications

Throttling and Debouncing	Limits the rate of function execution, reducing the number of calls	Optimizing the handling of user input, e.g., in search bars or during scrolling
React.lazy and Suspense	Delays the loading of non-critical components until they are needed	Improving the initial load time of the application by reducing the amount of JavaScript loaded
Immer	Simplifies working with immutable data structures, optimizing state updates	Scenarios with frequent and complex state updates, where there is a need to improve performance and code readability

2. Optimization tools

Today, there are a large number of tools that can make the development of React applications faster, easier, and more efficient, as well as optimize application performance.

Gatsby is a popular open-source static site generator based on React. This service allows developers to quickly develop their projects that will be simple and safe to use. Using integration with the extensive GraphQL plugin system and code sharing enables Gatsby.js to facilitate the rapid development of modern and fast web services.

Webpack is an open-source JavaScript module builder that is used to bundle code and associated resources into a single file. This tool also enables the use of Hot Module Replacement (HMR) in a React application.

Preact is a tool that is similar to React but differs in key ways in that it uses a more aggressive approach to optimization, relying on techniques such as memorization and lazy evaluation to minimize the amount of work that needs to be done to update the DOM.

Next.js is a tool for generating React applications and server code that simplifies their development and deployment.

Storybook is a tool that is designed for component libraries in various states. This tool can be used to develop a collection of components that can be viewed in isolation and tested independently of each other.

Razzle is a tool that abstracts away all the complex configuration required for applications by providing create-react-app, while still allowing developers to make other architectural choices.

Conclusion

Thus, the final choice of methods and tools for optimizing React applications depends only on the requirements and objectives of the project. Optimizing the performance of a React application can improve the user experience, reduce application failure rates, rank higher in search results, significantly reduce resource costs, reduce infrastructure requirements, and increase competitive advantage.

References

[1]. Thielens, T.M. React in action / T.M. Tielens. – St. Petersburg: Peter, 2019. – 368 p.
[2]. Vankevich, D.N. Comparison of frameworks for front-end development // D.N. Vankevich, N.V. Birch // Problems of development of science and education in the era of modernization. – Rostov-on-Don, 2023. – pp. 86-89.
[3]. Melikhova, K.A. Comparative analysis of state management technologies in React applications: Redux, Mobx and Reach context (Reach hooks) / K.A. Melikhova, E.S. Tomilin, A.S. Kropotov // XII Congress of Young Scientists. – SPb.: ITMO, 2023. – P. 395-402
[4]. Choosing the best framework [Electronic resource]. – Access mode: https://itanddigital.ru/frameworkit (Access date: 03/10/24)
[5]. Popular js frameworks [Electronic resource]. – Access mode: https://vc.ru/dev/147263-populyarnye-freymvorki-javascript (Access date: 03/10/24)