International Research Journal of Engineering and Technology (IRJET) e-ISSN: 2395-0056
Volume: 07 Issue: 04 | Apr 2020 www.irjet.net p-ISSN: 2395-0072

# Addressing the Limitations of React JS

**Vishal Gowda[1], Shanta Rangaswamy[2]**

*[1]Vishal Gowda, Dept. of CSE, R.V College of Engineering, Karnataka, India*
*[2]Shanta Rangaswamy, Associate Professor, Dept. of CSE, R.V College of Engineering, Karnataka, India*

---------------------------------------------------------------------***---------------------------------------------------------------------

**Abstract** - *React is a web framework that has better features compared to other similar frameworks such as Angular, Vue. This is due to its implementation of Virtual DOM; whose goal is to enhance the overall capability of the application. However, there are some things that should be kept in mind before designing applications which if ignored, the problems that may occur will lead to performance issues. Some of the commonly faced issues are component re-rendering, lag due to processing large data sets in a single stretch, application lag due to background computations being run, etc. This paper provides different ways to overcome such issues, thus enhancing the performance of React in a production environment.*

***Key Words***: React, State, Props, Components, Multithreading, Optimization, Performance, JavaScript.

## 1.INTRODUCTION

React is a web framework that was designed to address the performance issues in a web application. It uses virtual DOM which decides if the component must be reloaded based on the current state of the component and any changes that occur. This helps to prevent the application from re-rendering if not required. React has two-way data flow which controls the flow of the data inside the application which makes tracking easier and improves propagation and the stability.

The props and states of the component are two parameters that determine when a component should re-render in the application [1]. When there is a change or when a parent passes a new property to the child, the React DOM compares the new values to the previous values and re-renders only if there is a difference.

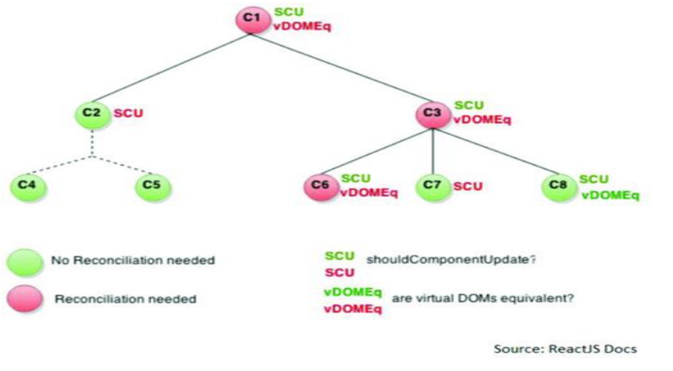

**Figure 1**: Component Tree for rendering a react component

Consider the hierarchy in Fig 1. For some change in the component C1, the component C1 decided to re-render. Now React checks the children in the subtrees of component C1 in a recursive manner until all the components in the subtree have been updated based on the value generated by the SCU() method.

The components in the tree are not forced to re-render by the parent component. It means that the children in that subtree will be evaluated. From the figure, it is notable that, since SCU of C1 is true, the subtrees C2 and C3 are checked. Since SCU of C3 is true, its children are evaluated, whereas the components in the other subtree C2 are not evaluated since SCU for C2 is false and similarly the process is continued. Finally, the components C6, C3, and C1 are re-rendered.

This type of re-rendering does not cause any issues in the case of a simple application, but in complex applications with many components, this re-rendering causes many performance issues.

And relying only on the React Virtual DOM comparison is not enough. Instead, a few additional measures are required to re-render the components only when required.

The following sections describe a few ways of improving the performance of React Applications.

## 2. OPTIMIZING THE REACT APP PERFORMANCE

### 2.1 Search Optimization

Searching large data sets involving millions of objects can consume a lot of time. If data received from the server is not in order, the time complexity will be O(n). Also, to create a single data structure of JSON objects causes a problem with memory overloading. Trying to build an application with 100,000 array objects may cause the application to fail. The NPM (Node Package Manager) throws a memory error. To fix this, data can be split into chunks of fixed sizes and use hashing, with the search attribute as the key.

In this case, the Hash generated using each element is used as the key to point to the Object [2]. Hence when there is a need to search the object based on its name, the Hash for the corresponding string is computed and used to find the object. The complexity for the search will be reduced to O (1).

It can be noted that since we have used hash technique, if data is searched using query string "user", then results will not be found. This is due to Hash("user") and Hash("user1") being different values. Hence, the computation of some more hashes by adding to the search string further is required.

For example, if data consists of string "user", then it is likely that strings such as "user1", "user2", "userx", etc. are present in the data set. Therefore, for the search string "user" building more such strings and looking up the corresponding Hashes in the Hashtable is a good solution.

The time complexity of these operations is k * O (1). Where k is the number of combinations of the search string. In comparison to O (n) and even O (n log n) approaches, this provides better performance [3].

If the data set consists of the following strings as Object attributes:

"user", "user1", "user2", . . ., "user22", . . .

Assume that the query string is "user".

The object corresponding to the hash can be retrieved, but performing searches in real time, there is also the need for retrieving related data as well. i.e. expecting the search to get the objects corresponding to strings "user1", "user2". But this is impossible with simple Hashing.

Building similar strings and compute the hashes to use them for quickly looking up objects. So, in this case, we can construct the following strings:

"user1", "user2", . . ., "userx"

And looking in the HashTable using the strings one at a time, when a string, say "userx" is not found in the HashTable, strings in a new form, say "userxx" can be taken into consideration.

In the same way, few other string combinations can be built. The number of combinations built determines the complexity of the algorithm. In sequential search technique, the time complexity will O (10^n) (assuming the length of the data set to be 10^n) whereas in this case, the time complexity is k * O (1), where k depends on the search string.

This is more like an absolute search, where data that resembles the search query most accurately is fetched, rather than fetching all data with slight resemblance. This is the tradeoff that reduces computation time [4].

But the main disadvantage is to fix the search attribute. Since the Hash Table is built using a specific object attribute, it is not possible to change unless we regenerate the HashTable using a different attribute. Although, if the search attribute is fixed, then this approach will be useful.

## 2.2 Utilize Existing Component Instances

For example, let's consider the size of the data set in the order of millions. Creating the instances at once will slow down the application.

To avoid this, a fixed number of instances are created and re-rendered with different Props whenever required.

For example, about 1000 objects can be created initially and whenever the user makes a request, references of the next 1000 objects are passed to the existing component instances [5]. It is recommended to pass the references to the object instances, as performing a copy of the objects will result in duplicating the instances and waste memory.

## 2.3 Reduce the number of State and Prop variables

This is an important measure to be taken. By reducing the number of State and Prop variables, the chances of the re-renders of the component that are not necessary can be reduced. Also, frequent and unnecessary changes to the State and the Props can be harmful to the performance of an application. The state must be updated only if it has a visual impact on the application and if not, the state must be updated only at the end [6].

The component could be stopped from rendering until the essential data has been received. This is configured by going over the SCU(props, state) method of the component. This method decides if the component should be rendered or not.

Consider a situation where a component receives data by making a REST API call to a server and a few local props received from the other components of the application. The component is rendered when it receives the server data. If data from the server is received before the props from other components, this would cause re-render. Instead, checks in the SCU() method can be added to verify the data, so that this problem can be successfully avoided. It helps to improve the application start-up render time and therefore improve performance [7].

## 2.4 Splitting the main component into individual components

In a code snippet, the component renders a table where clicking on any row or column would set that element to the state. This means that each time the cell is clicked, the entire table re-renders which causes performance problems. This is negligible if the size of the data used is small. But in real time applications, the size of data can be huge and rendering the entire table on every click is very inefficient [8].

A better way is by isolating each table row or column as an independent component and enable these components to listen and act on events independently. In this way, every time an event occurs, only one row or column will have to be re-rendered instead of re-rendering the entire table.

This can summarize be summarized as:

- Main Component

This component is responsible for creating and managing the child components and does not listen for all the events of the children unless the child sends a prop to the parent.

- Subcomponent

This is an independent component, that is made to handle all the occurring events, without having to interact with other components. Here, each row or column component will have its own state and every time there is a change, react virtual DOM will only compare the state for one particular row or column and re-render it if required. This is an improvement over the previous case, where an entire table was re-rendered.

Since React provides two-way data binding, an additional event is not required in the child element to publish the result back to the parent.

### 2.5 Multithreading

In most cases, web browsers spawn one thread per tab opened and this thread is responsible for all the operations performed in the application. Hence, if there are many computations to be performed, the thread would have to stop all other operations, and this leads to unresponsiveness in the application during this time [9].

But recently, Google and Mozilla have introduced Web workers to make browsers more powerful. A web worker is a JavaScript program which runs on a separate thread in parallel to the main thread therefore enabling us to implement multithreading in the application and use parallel method of execution [10].

Examples of operations that could be performed are:

- Data and web page caching.
- Image manipulation and encoding.
- Canvas drawing and image filtering.
- Network polling and web sockets.
- Background I/O operations.
- Video/Audio buffering and analysis.
- Virtual DOM diffing.
- Local database operations.
- Computationally intensive data operations.

Consider a case where there is a need for filtering the data in the application. If the size of data used by the application is small, multiple need not be running. But if the size of data is very large, then it is not efficient for a single thread to filter the entire data. In these cases, multiple web workers are created and assigned a specific portion of the entire data to each web worker. Each worker would now filter the portion of data assigned to it and store it in a specific space. When all the web workers complete their task, the main application thread just merges the pieces of the filtered data and obtains a result [11].

In the same way, web workers can be used to perform other background time-consuming tasks, while the application functions well without any performance degradation.

It is important to note that any time a web worker is created, it takes control of some system resources and these web workers last until the main web worker dies. So, it is important to make sure that we do not overkill by creating many web workers. Browsers such as Mozilla Firefox support up to 512 web workers running simultaneously.

## 3. CONCLUSIONS

React is a very useful framework having its own way of tacking performance issues that Web Applications commonly face. But in standalone applications, the performance can still be degraded when complex applications are required to be designed where the application deals with a lot of data processing [12]. This may lead to the application response issues and lag. This is a very common problem in large scale Enterprise Applications. This paper proposes some ways of tackling such problems within the web application.

The methods described are aimed at eliminating redundant computations and help in improving the performance of the application. While some techniques explained are specific to React framework, other techniques such as search optimization and multithreading can be implemented in any framework [13].

## REFERENCES

[1] ReactJS Docs reactjs.org/docs, “Optimizing Performance.” (2016).

[2] C. Wohlie et al., "Experimenting in Software Engineering" in, Springer, 2012. (pp. 1-10).

[3] S. Fröstl, Sebastian. "AngularJS performance tuning for long lists." tech. small-improvements.com (2013).

[4] Noam Elboim, blog.medium.com, "How to greatly improve your React App performance." (2018).

[5] Yu Yao, Jie Xia, "Analysis and Research on the performance Optimization of Web Application system in high Concurrency Environment" in, IEEE, 2016.

[6] Darrel Greenhill, Jack Francik, Jay Kiruthika, Souheil Khaddaj. “UX Design in Web applications: An approach to improve reuse in web applications”, In Web Engineering (pp. 335-352). Springer, Berlin, Heidelberg, 2016.

[7] A. Javeed, "Performance Optimization Techniques for ReactJS," (2019) IEEE International Conference on Electrical, Computer and Communication Technologies (ICECCT), Coimbatore, India, 2019, pp. 1-5.

[8] C. Ebert, M. Kuhrmann and R. Prikladnicki, "Global Software Engineering: An Industry Perspective," in IEEE Software, vol. 33, no. 1, pp. 105-108, Jan.-Feb. 2016.

[9] A. Singh, P. Chawla, K. Singh and A. K. Singh, "Formulating an MVC Framework for Web Development in React and Java “, 2018 2nd International Conference on Trends in Electronics and Informatics (ICOEI), Tirunelveli, 2018, pp. 926-929.

[10] Network, Mozilla Developer. "Web Workers API." (2015).

[11] Q. Yunrui, "Front-End and Back-End Separation for Warehouse Management System," 2018 11th International Conference on Intelligent Computation Technology and Automation (ICICTA), Changsha, 2018, pp. 204-208.

[12] H. Brito, A. Gomes, Á. Santos and J. Bernardino, "JavaScript in mobile applications: React native vs ionic vs NativeScript vs native development," 2018 13th

Iberian Conference on Information Systems and Technologies (CISTI), Caceres, 2018, pp. 1-6.

[13] S. Dhawan and R. Kumar, "Analyzing Performance of Web-Based Metrics for Evaluating Reliability and Maintainability of Hypermedia Applications," 2008 Third International Conference on Broadband Communications, Information Technology & Biomedical Applications, Gauteng, 2008, pp. 376-383.