International Journal of Applied Engineering Research
ISSN 0973-4562 Volume 10, Number 9 (2015) pp. 22939-22946

http://www.ripublication.com

Proposal of Browser By Exploiting The Proficiency of V8 For Large Data Visualization

Shiju Sathyadevan [1], Jaison P.J [2], Geethu Sankar [3]
[1] *Assistant Professor, Amrita Centre for cyber security, Amrita University, Kerala, India*
[2] *PG Student, Department of Computer Application, Amrita University, Kerala, India*
[3] *PG Student, Department of Computer Application, Amrita University, Kerala, India*

Abstract

Big data is a commonly discussed topic of today. The 4V's (volume, velocity, variety and veracity) of big data is creating hurdles in many areas which also includes browser. Loading data in dynamic manner is hurdle faced by the browser. The performance falls down as the magnitude of data to be processed escalate. When the data size increases above a certain level it throws error in the java script followed by the crash of the browser. The performance of different browsers varies depending on the techniques they employ. Here we propose a solution based on Google V8 engine for handling big data by testing its efficiency in standalone mode.

Key Words: Google V8 Engine, JSON, Big data, Node WebKit, JavaScript Engine, Visualization.

Introduction

Big data is a technical term used for the data which cannot be managed or processed properly using the traditional methods. Large amount of data are being created every seconds from different sources which consists of a lot of information useful in many aspects [7]. Today Big data is only dynamically processed using stand-alone applications. Since the insistence of big data processing is high, the applications to process it are in very much in demand. The cost of these applications makes it heavy for a common man's budget. A free source web browser for big data processing will be a widely acceptable approach.

In browser perspective the data which can be dynamically handled and processed with java script is only 5 MB (results obtained from our test) .Since the data to be processed is very high and the ability being very low this increases the complexity at the part of the browser. The large amount of data provided to a browser can degrade its performance. As the data size increase the browser runs into errors with scripts and

it further leads to the crash of the browser. To solve this issue we are proposing a modification to the existing browser based on Google V8 as an open source rendering engine.

The paper is divided into four sections. The first segment mainly discusses about the current browser issues, limitations and the test methodology adopted to test the efficiency of the browsers. The second section discuss on the V8 architecture and testing its efficiency in standalone manner. The testing is conducted using two methods. First with a program to load data into terminal and the second using node Webkit (nw.js) .The third section deals with the system proposed for handling big data.

Existing System Issues

The browsers are considered as software with low processing capability. They are lacking the necessary feature of handling huge data which is considered as an essential factor for software in today's world.

The most commonly used open source web browsers today are Google chrome, Mozilla Firefox, Opera and Safari. Each of them have their own features which make them different. They differ each other in their performance. The java script rendering engine used by them contributes to their performance [10]. The Google chrome and Opera uses V8 while Mozilla Firefox uses Spider monkey and Safari uses Nitro as their rendering engine. We conducted a test to evaluate the JavaScript efficiency of each browser.

Test Methodology

The browsers were tested on the efficiency to visualize the bubble chart of d3.js across different JSON data size.D3.js is a Java script library that creates and manages interactive and dynamic graphs with digital data [12].

The data format that we have used is JSON. It is the widely used data -interchange format. The main features of it are light weight, Language independent, self-describing and easy to understand.

The Syntax of A JSON File

```
{
object { [  //array
        {"key":"value"},    //object 1
        {"key":"value"},    //object 2
        {"key":"value"}     //object 3
      ]}
}
```

The Sample of The JSON File Used For Testing

```
{
 "name": "flare",
 "children": [
```

```
{
 "name": "analytics",
 "children": [
  {
   "name": "cluster",
   "children": [
    {"name": "jklCluster", "size": 3938},
    {"name": "CStructure", "size": 3812},
        {"name": "MergekingEdge", "size": 743}      ]
  },
  { [.....
  ]}
}
```

We have used Google chrome version-39,Mozilla Firefox version 35,opera version 12.16 in Ubuntu 12.04(64-bit operating system) and safari version5.1.7 in Windows7(64 bit) for testing .

The test result of each browser is provided below

Data Size/Browser	Google chrome(V8)	Firefox (Spidermonkey)	Opera (V8)	Safari (Nitro)
10kb	✓	✓	✓	✓
100kb	✓	✓	✓	✓
150kb	✓	✓	✓	✓
300kb	✓	✓	✓	✓
400kb	✓	✓	✓	✓
500kb	✓	☑	✓	✓
750kb	✓	☑	✓	☑
850kb	✓	×	✓	☑
1mb	✓	×	✓	×
2mb	✓	×	✓	×
3mb	✓	×	✓	×
4mb	☑	×	☑	×
4.5mb	☑	×	☑	×
4.95mb	×	×	×	×

Figure 1: Test result table

✓ - Successful visualization of data.

☑ - Performance Lagging

× - Crashed

Fig:1 shows the detailed output of the test conducted. This shows that the browsers are only able to handle below 5 MB of JSON data dynamically, which is very low compared to today's requirement. Even though they consume a large part of the system resources such as memory and the CPU their performance is very low. This paves the way to researches in order to increase the data capacity and to decrease the requirement availability gap.

The JavaScript rendering engine plays a key role in the performance of each browser [9]. The Mozilla Firefox uses Spider monkey whose performance is low compared to the Google's V8 engine which is used both by chrome and opera. The main difference being projected for the performance gap that is the spider monkey compiles the code into intermediate code and then to machine dependent code with the help of interpreter, but in Goggle V8 the JavaScript is compiled directly to machine dependent code [6]. The web kit used in safari is also working in the same manner but the memory leak problem is very high .The automatic garbage collection and hidden classes features make V8 flexible and adaptable to the real time situations. [3]

The other feature that contributes to the performance of the goggle chrome is multi process architecture [8]. A new process is started when a new tab is opened which helps in handling the process very effectively. The plug-in and the extensions are also working as different process. On the other hand the Firefox employs the single process architecture, which degrades the browser performance. The above cited features of V8 prompted us choose it for creating a solution for the problem.

Google V8 Java Script Engine

The V8 is a combination of JavaScript and C++. It allows C++ application to reveal its objects and function to a JavaScript [1]. We can use C++ code for input output operations and java script is used for creating functions. We can use both internal and external java script function in a V8 program. If the function is internal we can directly compile the script and execute it [5]. If we are using a function written in an external JavaScript file, the function is loaded into the V8 program in the form of a string and compiled to a script in the main program using V8.

Evaluating Efficiency of V8

i. Using V8 program to load data into terminal

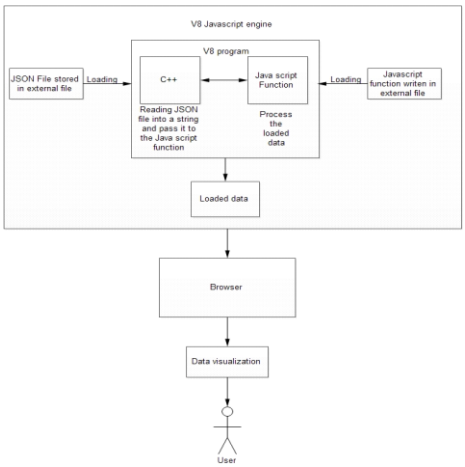

Figure 2: Google V8 Architecture Diagram

The fig: 2 above explain the working of the V8 program written to check the efficiency of V8 to load the big data. The JSON file is loaded into the V8 program using the C++ code. The content of the file is stored into a V8 String variable .The JavaScript function written in an external JavaScript file is loaded into the V8 program and the function is executed by passing this variable as its argument. The function returns the JSON data to the main V8 program, which is displayed through a terminal.

A Sample JSON file of 340 MB (With limited resource of 4 Giga Bytes of RAM and Core I5 Processor) was created using a java program and loaded into V8 JavaScript function. In a standalone system V8 loaded 340 MB data successfully. It is proved that it can handle up to 1 Giga bytes of data in a standalone mode.

Fig: 3 show the test result of loading 340 MB data into the terminal using the V8 program that we have developed.

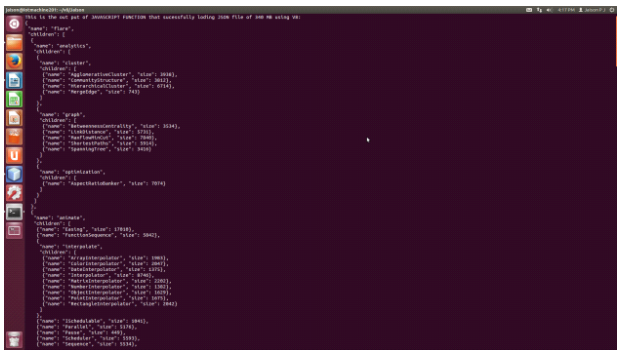

Figure 3: Screen shot of loading 340 MB data using a V8 program.

ii. Using Node - Web Kit (nw.js)

Node-Webkit is a runtime application based on node.js (V8), Chromium and it is developed by Intel at the Intel Open Source Technology Canter [2]. Fig: 4 explain the architecture of Node Webkit application used. We can write applications in JavaScript and HTML with Node-Webkit. It helps you to call Node.js modules directly from the DOM and enables you to create a new trend of writing applications with the help of Web technologies [4].

The library combines Node.js and Webkit engine in a particular way. Both Node and Webkit share the same background, allowing developers to write code in a same way in which it has to be executed in a browser, but with the addition of all Node's features.

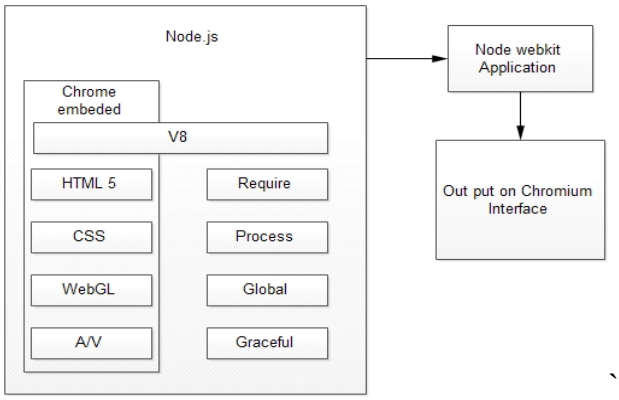

Figure 4: Node Webkit Application Architecture.

As an alternative method for testing V8 efficiency, we created a Node Webkit (nw.js) application for loading JSON file. It was able of loading more JSON data to the D3.js bubble chart. It was almost more than double the data limit of Google chrome web browser.

The performance of Node Webkit application proves that it is more efficient than the Google Chrome browser since it loaded more than 10 MB of JSON data using only 30 percentage CPU and 40 percentage memory in 70 seconds of time to visualize in the Chromium interface. The performance evaluation was done on a system of Intel I5 processor with 4 Giga bytes of RAM. Google chrome browser On the other hand used 50 percentage of memory with 100 percentage of CPU usage to load about 4.95 MegaByte of JSON data to the same D3.js Visualization bubble chart in 90 seconds.

The reason for high performance of node web kit (nw.js) is because of its access to more system resources like the system cache and host operating system. It is having the capability to increase the local storage limit.

Proposed System

The Google V8 engine is having a capability of handling huge data with the available resources. It was able to handle about 350MB (From our test we conducted) of data with a resource availability of 4 Giga Bytes of RAM and Core I5 Processor. This states the efficiency of V8 in handling data so a browser built on V8 should be capable of handling data which should more than the current capability with necessary modification.

The browsers are given limited access to the resources. Sandboxing shrink the efficiency of the browsers when compared to other standalone applications employing the similar technique. There are components in browser which consume memory other than the tab processing like the layout engine, the Java script engine, the tab frame works. The big data handling and visualization require a parsing of large JSON objects .Since a browser developed using V8 which is having more java script efficiency compared to other browser ,providing it with more system resources can make it more capable of handling and processing big data.

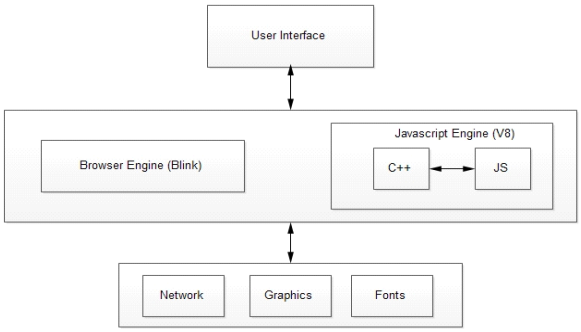

Figure 5: Proposed System Architecture

The V8 engine work alongside blink which is the layout engine used in chrome. The V8 will perform the java script compilation and execution part[1]. The C++ objects and function will be exposed to the java script in the V8 [3].The java script functions can be internal or external files. When an event is triggered the JavaScript will manipulate the DOM objects .The blink does the graphical user interface part. It includes painting and the layouts of the DOM objects. The browser will be having more modified features like more access to the memory, which will help in the speedy processing of the objects.

In our proposed system, the memory allocation is based on the available memory rather than having a fixed memory limit for the tab. If a tab requires more memory for loading and processing huge data it will be prioritized as high priority process and provide it with more memory. This will be backed by the V8 efficiency. If the full capability of the V8 is utilized in the browser the browser will be very effective to handle and visualize large data.

Conclusion

The paper affirms the use of V8 for handling big data for a browser. The test performed by us shows the efficiency of V8 to handle data with the limited available resources. Providing browser with more access to the system resources like memory and removing the per tab memory limit backed with V8 java script efficiency will make it powerful and scalable in big data handling .More researches have to be conducted in this area. The proposed browser will to narrow down the gap between demand and the actual performance of the browser.

Reference

[1] https://code.google.com/p/v8/
[2] https://github.com/
[3] https://developers.google.com/v8/
[4] http://stackoverflow.com/

[5] Jan Kasper Martinsen, Håkan Grahn and Anders Isberg, Combining Thread-Level Speculation and Just-In-Time Compilation in Google's V8 JavaScript Engine

[6] Antti P Miettinen, Software Techniques Just-in-time compilation JavaScript Engines ,April 25, 2010

[7] Pavan Sridhar, Neha Dharmaji, A Comparative Study on How Big Data is Scaling Business Intelligence and Analytics

[8] Adam Barth UC Berkeley, Collin Jackson,The Security Architecture of the Chromium Browser

[9] Alan Grosskurth , Michael and W. Godfrey, A Reference Architecture for Web Browsers

[10] Duan Hucai , Ni Hong, Deng Feng and Hu Linlin,A Parallelization Design of JavaScript Execution Engine

[11] Jan Kasper Martinsen and Håkan Grahn Anders Isberg, A Limit Study of Thread-Level Speculation in JavaScript Engines — Initial Results.

[12] http://www.w3schools.com/json/default.asp