Ⅰ. Introduction

Mobile web App runs on mobile browser. The browser only hosts the application presentation layer that is designed using HTML5. For enterprises, a commensurate large portion for mobile applications is based on making existing desktop or web-based applications available on mobile devices[1].

one of the main challenges associated with building mobile applications is selecting the appropriate architecture capable enough to support a large and increasing number of clients requests handled by a Client–Server architecture simultaneously. However, web server based on the thread-based approach
might perform inefficiently as the number of incoming network requests increases. That is the reason that many industry such as eBay, LinkedIn have started to adopt event-driven programming as an option to respond to a large number of concurrent requests and achieve scalability more operationally [2].

The main contribution of this work is to investigate the performance of Client–Server Architecture that includes backend server, web programming framework and database able to support a large and increasing number of concurrent clients’ requests. The performance metrics are throughput, response time and error rate to compare web applications developed using JavaScript and JavaServelet.

II. Node.js platform for concurrent connections

One of the advantage of Node.js over thread-based framework is that Node.js has a built-in single-thread event loop and non-blocking model [4]. The second advantage is that Node.js allows event-driven paradigm. This event-driven paradigm is the key on which interactive Node.js applications are constructed. Node.js features the event-handler that creates events and the main loop executes the appropriate event. The event handlers in Node.js are known as callback functions. Therefore, callback functions are eventually executed when the non-blocking operation completes. So, when the event loop in Node.js receives the completion feedback, it executes the callbacks.

The figure 1 shows the model structure of the Node.js platform [3]. At the core of Node.js, we have the event loop running in a single process and in infinite manner. This means, that the event loop concepts looks continually at what events have submitted and what callbacks need to be executed. This architecture ascribes Node.js a high level of concurrency and therefore higher overall throughput.

III. Implementation of a Remote On-line Diagnosis Service based mobile Apps

For this study, we use an On-Board Diagnostic simulator called OBD-II simulator (ECUsim2000)[5]. With this ECUsim2000 simulator, communication test is conducted in the same way as an actual car was developed and tested. The hardware architecture includes the ECUsim 2000 OBD-II ECU simulator for reading the performance using OBD-PIDs code make up in the android application such as speed, Revolutions Per Minute (RPM), Intake Temperature, Coolant Temperature. The proposed system architecture includes Bluetooth communication between the ECU simulator (ECUsim 2000 and Bluetooth interface) that supports all OBD-II protocols and receiver (mobile) devices. Figure 2 shows a screen shot...
while the car owner monitors on-board diagnostics saved on the MongoDB database.

IV. Simulation and Experimentation results

To measure the performance of different use cases the program Apache JMeter 2.712 was used [6]. The component under test was the main back end server. The simulation of client–server architecture is presented on the Figure 3. It shows a Node.js request to write and query data from database.

In this paper, we have considered three kind of Client–Server Architecture that provides backend for server–side implementation and database layers. The first Client–Server Architecture is that the server–side code resides on the Node.js web server and the database is MongoDB[7]. The second Client–Server Architecture is that the server–side code resides on the Apache tomcat server. The application server that implements the http request is writing using JavaServer Pages (JSP) technology [8]. JSP uses the Java programming language. With this model, a relational database MySQL is used as the database. The third Client–Server Architecture consists of Apache Tomcat on the server–side and MongoDB database. Here, we have used the Java API for MongoDB/BSON in Apache Tomcat [9]. The web service functions on Node.js would collect data from the client system such as web service or web-based application on smartphone. The peak load testing scenarios state is shown in the table 1

<table>
<thead>
<tr>
<th>Scenario</th>
<th>Concurrent users</th>
<th>Ramp-up period(s)</th>
<th>Loop times to run the similar sample</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scenario 1</td>
<td>200</td>
<td>0</td>
<td>50</td>
</tr>
<tr>
<td>Scenario 2</td>
<td>1000</td>
<td>0</td>
<td>50</td>
</tr>
<tr>
<td>Scenario 3</td>
<td>2000</td>
<td>0</td>
<td>50</td>
</tr>
</tbody>
</table>

The Figure 4 shows the results of the three Client–Server Architecture configuration. To this performance, we analyze the throughput and response time metrics. The Node.js–JavaScript–MongoDB configuration outperforms.

For this architecture, from 200 up to 2000 concurrent users (from 10000 to 100000 requests), the response time is high within 173ms but the throughput in comparison to the response time is less low with 164 requests per second. This signifies that this Client–Server Architecture is capable enough to sustain a large number of concurrent clients’ requests. The Apache Tomcat–JSP–MySQL has a higher response time but the throughput is much lower within 117 requests per second. This signifies that this Client–Server Architecture is not capable enough to execute concurrent requests. The third model that include Apache at server–side and MongoDB as database outperforms less better in comparison to
Node.js—JavaScript—MongoDB but better than Apache Tomcat—MySQL. Therefore, Node.JS is roughly 40% faster, for example 164 responses per second against 117ms for 2000 users that corresponds to one hundred thousand (100000) concurrent requests.

V. Conclusions and Future Work

The Client–Server Architecture constitute of Node.JS server side and MongoDB is 40% faster that the Java EE solution using Apache Tomcat at the server side with MySQL or MongoDB database for implementing mobile client server computing applications. In this paper, the difference concurrency models between single-threaded event loop Node.js and multi-thread approach made difference. To test Node.js a higher concurrency level—where it is supposed to surpass multi-threading, other problems like increasing the number of requests occur. The reason is that Apache JMeter is a 100% pure Java application to evaluate the functional behavior and measure performance of the three Client–Server Architecture configuration. We were not able to run these tests beyond 4000 concurrent users, what means over 200000 requests. For future work, we will look the impact of using the node.js in a real time remote and controlling IoT application such as Home automation.

References