International Journal of Advanced Culture Technology Vol.4 No.3 13-19 (2016)
http://dx.doi.org/10.17703/IJACT.2016.4.3.13

IJACT 16-3-3

Performance Evaluation of Node.js for Web Service Gateway in IoT Remote Monitoring Applications

Lionel Nkenyereye, Jong-Wook Jang*

Department of Computer Engineering, Dong-Eui University, Busan, Korea
*Lionelnk82@gmail.com, jwjang@deu.ac.kr**

Abstract

The growth of mobile devices in Internet of Things (IoT) leads to a number of remote and controlling system related IoT applications. For instance, home automation controlling system uses client system such web apps on smartphone or web service to access the home server by sending control commands. The home server receives the command, then controls for instance the light system. The web service gateway responsible for handling clients' requests attests an internet latency when an increasing number of end users requests submit toward it. Therefore, this web service gateway fails to detect several commands, slows down predefined actions which should be performed without human intervention. In this paper, we investigate the performance of a web server-side platgorm based event-driven, non-blocking approach called Node.js against traditional thread-based server side approach to handle a large number of client requests simultaneously for remote and controlling system in IoT remote monitoring applications. The Node.JS is 40% faster than the traditional web server side features thread-based approach. The use of Node.js server-side handles a large number of clients' requests, then therefore, reduces delay in performing predefined actions automatically in IoT environment.

Keywords: *web service gateway, Event-driven approach, Node.js, remote monitoring, concurrent programming, Internet of Things, performance.*

1. Introduction

Traditionally, the remote and controlling system use client server architecture such as web-based mobile application or web service gateway to handle commands to control devices in (Internet of Things)IoT environment [1]. As the number of clients' requests increases in simultaneously manner, the web service gateway to handle those requests encounters limitations for performing them successfully, and therefore, slows down predefined actions that take automatically without the help of humans.

The architecture of web server-side and its scripting approach on the gateway implementing web service must inherit features that allow to respond to an increasing number of network requests from the end-users. However, web service gateway based on the thread-based approach might perform inefficiently as the

Manuscript Received: Jul. 4, 2016 / Revised: Jul. 15, 2016 / Accepted: Jul. 28, 2016
Corresponding Author: jwjang@deu.ac.kr
Tel: +82-51-890-1709, Fax: +82-51-890-1724
Department of Computer Engineering, Dong-Eui University

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. The architecture implements a web service gateway which is 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.

2. Event-driven server side concept features Node.js platform

The architecture of the web server features event-driven improve scalability more efficiency for handling several requests simultaneously .The aim of adopting event-driven server-side architecture is to prevent blocking and long running requests which may increase cost, consequently let out users to several unlike convenience use of their smart phone applications. Certainly, the adoption of an event-driven web architecture to build web-based mobile application would enhance the desire of end users that use remote monitoring applications based on IoT to safely access their smart phones applications at the same time enable commands controlling connected devices on mobile application running in background such remotely switch power on/off, get information of device's power consumption and ambient temperature, and eventually control the nearby devices using the integrated IR-emitter[1].

The Node.js is a platform for building event-driven networking programs. It is a version of Google's V8 JavaScript. It runs in a single thread that enables it to serve many clients concurrently. The Fig. 1 shows the model structure of the Node.JS platform [4]. 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.

As shown in Fig. 1, the event loop queues both new requests and blocked I/O requests. The single-thread executes an event loop by setting up a simple mapping of all requests. The event loop gradually dequeues requests from the queue, then processes the request, and finally takes up the next request or waits for new requests.

Using of Node.js makes it possible to write scalable network applications that inherit concurrency in their design. This is accomplished by event-loops. To avoid blocking, all main Application Programming Interface (API)-calls that involve Input/ Output are rendered asynchronous. Thus, instead of waiting for the function to return, the event-loop can execute the next event in the queue; and when the API-call is finished it activates a callback function that specifies when the call was made. There is an infinite loop running in a single process that continually monitors what event occurred and what callbacks need to be executed. It deals with queuing all events automatically and keeps making the appropriate callbacks as fast as it can. A developer using Node.js does not really need to know this, though they need to know that the callbacks will be activated as quickly as possible when events occur and the task completes [2][3].

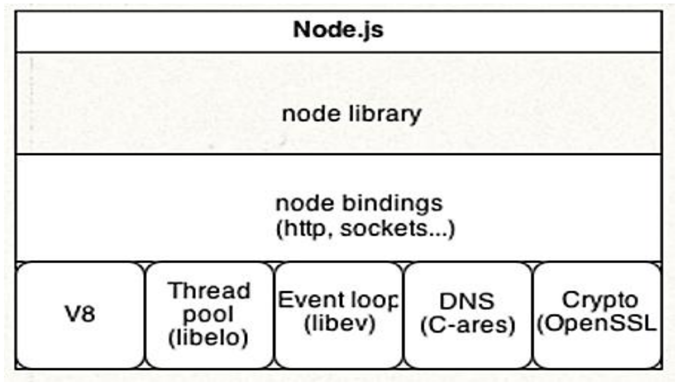


Figure 1. The structure of Node.js. The single thread handles all incoming requests. The event loop running on the single thread continue look at what events have occurred and what callbacks needed to be executed. All components are single thread in way these components interacts in asynchronous manners, then able to interface with one another. Node runs on Google V8 (source [4])

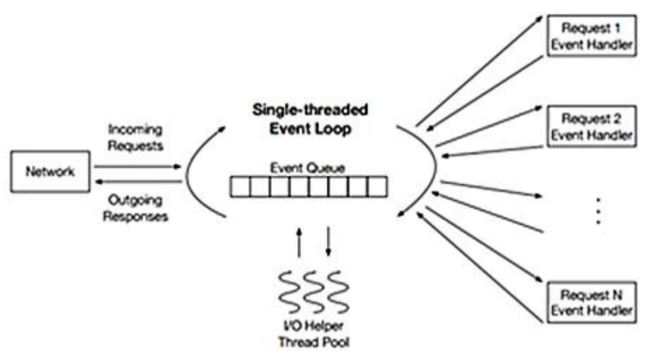


Figure 2. Event-driven server-side for web service. This is a Conceptual Model for an event-driven architecture. Each incoming client request is handled by the single-thread event loop. Event handlers do trigger I/O actions that result on a new event later asynchronously.

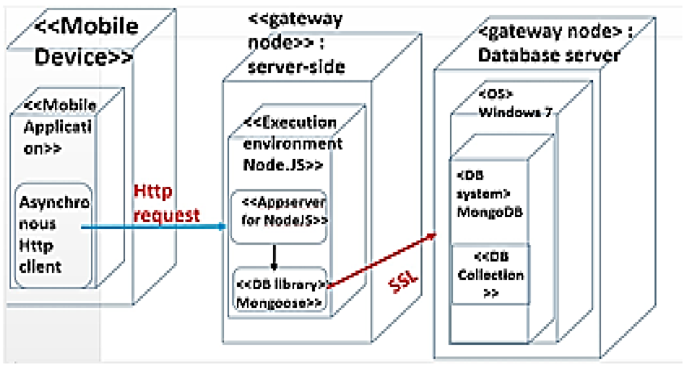


Figure 3. A deployment diagram of a Node.js system. It depicts Asynchronous Http client and communicate with the Node.js server application running on windows server.

3. Performance Investigation of server-side framework based Event-driven against no JavaScript server side implementing web service gateway

To measure the performance of different use cases the program Apache JMeter 2.712 was used [5]. The component under test was the main back end server. The simulation of client-server architecture is presented on the Fig. 3. It shows a Node.Js request to write and query data from database.

The component under test was the main back end server. JMeter works by simulating multiple users making multiple requests to the server. Every user is run in a separate thread. To simulate normal conditions, JMeter allows a “ramp-up” time to be specified. The ramp-up is the time it takes to “ramp-up” the full number of threads chosen. Several tests of each scenario were carried out under different cluster of web service gateway configurations and varying amount of loads generated by Apache JMeter. To simulate normal scenario, JMeter allows to configure a specific number of users and a “ramp-up” time to be specified. The ramp-up defines the virtual users’ arrival rate. Hence, JMeter prompts to start all virtual users defined within the time specifies in “Ramp-up period”. For instance, if we have 100 threads (users) and 100 seconds ramp-up. This means that JMeter will start user 1 and after every 1 second, it add 1 more user.

The test plan use JMeter to capture throughput, response time results of Node.Js‘s single-thread event-loop against the traditional application based Java on the Apache Tomcat. The capacity and performance testing is required to show that an web service gateway for remote monitoring related IoT applications consists of backend and database layers can run with acceptable responsiveness when a large number of concurrent users can access the backend server-side and database simultaneously.

In this study, we have considered three kind of architecture to implement web service gateway. Each of this architecture provides backend for server-side implementation and database layers. The first architecture for web service gateway integrates a server-side code resides on the Node.JS web server and the database MongoDB for our case (see Fig. 3). Thus, MongoDB fits perfectly for Node.jS applications. Therefore Node.js and MongoDB letting us write JavaScript for the backend and database layer [6]. Furthermore, MongoDB is known for its schemaless nature gives a better way to match the constantly evolving data structures in remote monitoring related IoT applications. The second architecture integrates server-side code resides on the Apache tomcat server. The application server that implements the http request is writing using JavaServer Pages (JSP) technology [7]. 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 [7]. For each of the architecture for web service gateway, the goal is not to test the web application but to listen to the http request sent from the mobile device in the same way an http request is submitted from a web browser. The Application under test is based on the mobile client server computing that has a module of sending the commands of control from mobile device based remote monitoring IoT apps to the web service gateway.

The test environment for the first architecture for web service gateway consists of web framework for Node.js, Node.js server-side for backend and MongoDB for database. This test model environment is configured with the architecture outlined below:

(i) A user http request arrives over a SSL to the application server
(ii) The application server forward calls to the server in the web tier
(iii) The web tier runs in a computer 3.30 Ghz Intel core i3, 8GHz of on windows server
(iv) The web tier runs Node.JS server, Express for Node.Js and code of MongoDB object modeling for node.js
(v) The data tier runs on a separate single virtual server, which hosts the MongoDB database.

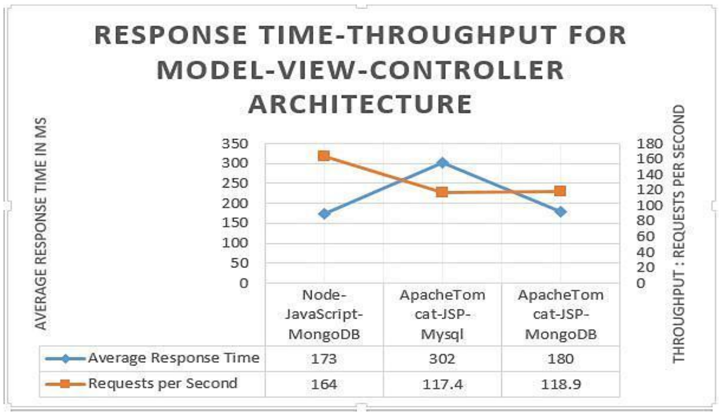
The rest of the Client-Server architecture have the same environment as the first except both the web tier and data tier configuration

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 1. Scenarios cases for experimentation

#scenario	Concurrent users	Loop (times to run the similar sample)
Scenario 1	**200**	50
Scenario 2	**1000**	50
Scenario 3	**2000**	50

3.1 Discussions of the results



	Node-JavaScript-MongoDB	ApacheTomcat-JSP-Mysql	ApacheTomcat-JSP-MongoDB
Average Response Time	173	302	180
Requests per Second	164	117.4	118.9

Figure 4. The performance of the three model architecture of web service gateway. The measurement metrics are throughput and response time

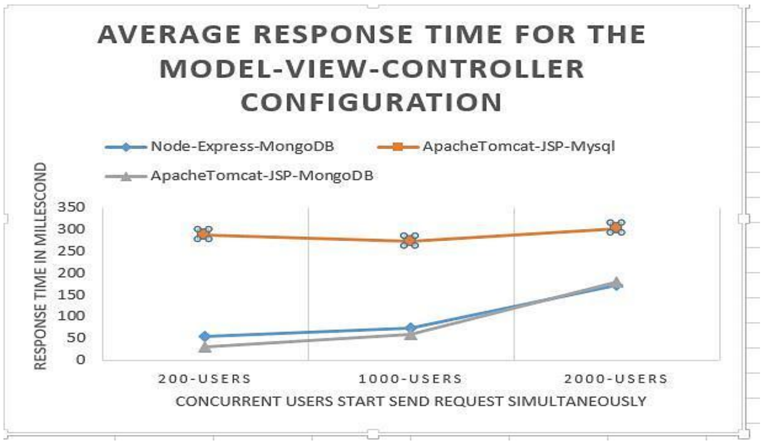


Figure 5. The measurement of the response time on average for the three-model architecture of web service gateway as the number of concurrent users increase

The Fig. 4 shows the results of the three configuration applied to the architecture of web service gateway. 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 architecture of web service gateway 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 architecture of web service gateway 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. What we can see on Fig. 5 that the response time degrades as the number concurrent requests increases. For example, Node.JS-MongoDB was within response time of 54ms on average at 200 concurrent users, and 173ms on average at 2000 concurrent requests. We can see that for Apache-Tomcat at the server side, the average response time has an almost linear correlation to the number of concurrent requests. This means that a thousandfold increases in concurrent users starts to a hundredfold increase in response time. This generates to formulate that the number of concurrent users carried out by an Apache-Tomcat at server-side is not relatively constant. 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.

5. Conclusions and Directions for Future Work

The architecture of web service gateway constitutes 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 architecture of web service gateway. 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.

Acknowledgement

This research was supported by the Brain Busan 21 Project (2016), Nurimaru R&BD project (Busan IT Industry Promotion Agency) (2016), and Dong-Eui University Research Institute (2016).
This paper is a revised and expanded version of a paper entitled Performance Investigation of Event-Driven Server-Side for Internet of Things based Remote Monitoring Applications presented at 2nd International Integrated (web & Offline) Conference & Concert on Convergence, Saint Petersburg State University, August 7-14, 2016.

References

[1] L. Wang, D. Peng, and T. Zhang “Design of Smart Home System Based on WiFi Smart Plug,” The International Journal of Smart Home(IJSH), Vol. 9, No. 6, pp. 173-182, 2015.

[2] Yuhao,Z.,Daniel,R.,Matthew,H.,Vijay,J.R., “Microarchitectural implications of event-driven server-side web applications”, Proceedings of the 48th International Symposium on Microarchitecture, pp: 762-774, 2015.

[3] S. Tilkov, S. Vinoski, “Node.js : Using Javascript to Build High-Performance Network Programs”. Internet Computing, IEEE, 2010 STRIEGEL, GRAD OS F’11, PROJECT DRAFT 6.

[4] S.Benjamin, L.Maude. “An Inside Look at the Architectural of NodeJS”,http://mcgill-csus.github.io/student_projects/Submission2.pdf

[5] H. Emily , “Apache JMeter. A practical beginner’s guide to automated testing and performance measurement for your websites, PACKT PUBLISHING, BIRMINGHAM-MUMBAI, 1-138, 2008.

[6] K. Brian, “CS764 Project Report: Adventures in Moodle Performance Analysis”, http://pages.cs.wisc.edu/~bpkroth/cs764/bpkroth_cs764_project_report.pdf, pp:1-28, 2016

[7] L.N. Glenn, “Tomcat Performance Tuning and Troubleshooting”,ApacheConference, pp:1-10,2003