

The 6th International Conference on Current and Future Trends of Information and
Communication Technologies in Healthcare (ICTH 2016)

Performance Evaluation of Server-side JavaScript for Healthcare Hub Server in Remote Healthcare Monitoring System

Lionel Nkenyereye, Jong-Wook Jang^{a*}

Dong-Eui University, 176 Eomgwangro, Busanjin-Gu, Busan, 614-714, Korea

Abstract

With the help of a small wearable device, patients reside in an isolated village need constant monitoring which may increase access to care and decrease healthcare delivery cost. As the number of patients' requests increases in simultaneously manner, the healthcare hub server located in the village hall encounters limitations for performing them successfully and concurrently. In this paper, we propose the design tasks of the Remote Healthcare Monitoring application for handling concurrency tasks. In the procedure of designing tasks, concurrency is best understood by employing multiple levels of abstraction. The way that is eminently to accomplish concurrency is to build an object-oriented environment with support for messages passing between concurrent objects. Node.js, a cross-platform runtime environment features technologies for handling concurrency issue efficiently for Remote Healthcare Monitoring System. The experiments results show that server-side JavaScript with Node.js and MongoDB as database is 40% faster than Apache Sling. With Node.js developers can build a high-performance, asynchronous, event-driven healthcare hub server to handle an increasing number of concurrent connections for Remote Healthcare Monitoring System in an isolated village with no access to local medical care.

© 2016 Published by Elsevier B.V. This is an open access article under the CC BY-NC-ND license
(<http://creativecommons.org/licenses/by-nc-nd/4.0/>).

Peer-review under responsibility of the Program Chairs

Keywords: concurrent application; asynchronous I/O; server-side JavaScript; healthcare hub server; Internet of Things; remote healthcare monitoring system; Node.js; web services;

* Corresponding author. Tel.: +821056000845; fax: +82518901724.

E-mail address: jwjang@deu.ac.kr

1. Introduction

The idea of devices connected with each other leads to an enormous systems based Internet of Things (IoT) such as IoT-related healthcare systems. The implementation of these IoT-related healthcare is based on the definition of the IoT as a network of devices connected with each other in order to capture and share useful data through wireless connectivity and Internet access capabilities that connect to a central processing and controlling server in the cloud¹.

Remote Healthcare Monitoring (RHM) is a technology to enable monitoring of patients reside in an isolated village with no access to effective health monitoring of conventional clinical settings². However, most of RHM technologies follow a general architecture that consists of four components². First component is variety of sensors on a device that is enabled by wireless communications to measure physiological parameters. The second component is a local data storage on the coordinator mobile device (smartphone) at patient's site that interfaces between sensors and centralized data repository and healthcare monitoring hub server. The coordinator device is equipped by wireless connectivity such as Wi-Fi or 3G/4G LTE enabled internet to send collected physiological data to healthcare providers. The third component is a centralized repository to store data sent from sensors, local data storage, diagnostic application, and healthcare providers. The last component is the diagnostic application software that send a real-time alert to the patient based on the analysis of collected data by triggering a series of actions previously agreed upon by the patient and medical practitioners. In this paper, we use the term healthcare hub monitoring server for referring to the third and fourth component. The Fig.1. shows the architecture for RHM system.

The healthcare hub monitoring server of the RHM system aims to handle a large number of requests from patients in a concurrency manner at the point to handle treatment recommendations efficiently. Concurrency is the tendency of things that happen at the same time in a system. Concurrency is particularly important in RHM system that have to react to outside events in real-time and that often have hard deadlines to meet³. As the number of patients' requests increases in simultaneously manner, the healthcare hub server located at the village hall encounters limitations for performing them successfully. Therefore, this RHM application fails to detect deterioration of patient's health, slows down predefined actions previously agreed upon by the patient and the doctor. As consequences, number of emergency department visits increases, hospitalization and duration of hospital stays increase healthcare delivery costs. In addition, patient cannot receive real-time alerts which are a series of actions that take place automatically without the help of doctors. These alerts remind the patient to take some precautions for instance. This situation can cause the death of patient when the doctor cannot receive new data and the patient's history from the healthcare hub server. Furthermore, the patient should not benefit any immediate medical attention.

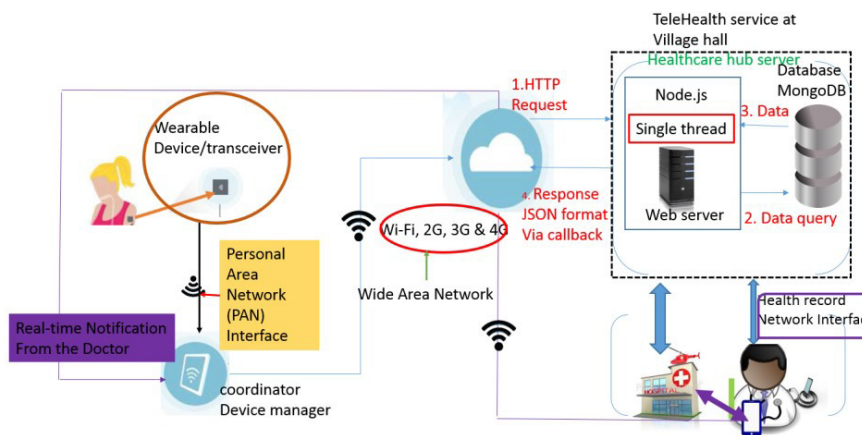


Fig. 1. Architecture of Remote healthcare monitoring system in an isolated location with no access to local medical care.

When we talk about concurrency, two approaches are proposed⁴. The first approach is to have multiple threads in a single process. The second approach is to have multiple single-threaded processes. Using separate processes for concurrency also has an additional advantage. The advantage is to run the separate processes on distinct machines connected over the network. Though this increases the communication cost. The alternative approach is to run multiple threads in a single process. Each thread runs independently of the others, and each thread may run a different sequence of instructions. The disadvantage of the multiple threads approach is that all the threads in a single process share the same resources. In case the shared resource fails, the active threads included in the single process are destroyed as well.

In this paper, we propose the design tasks of the RHM application for handling concurrency tasks. In the procedure of designing tasks, concurrency is best understood by employing multiple levels of abstraction⁵. First, the functional requirement of the RHM system must be well understood in terms of its desired behavior. Next, possible functions for concurrency should be checked into. This is best done using the abstraction of threads. The way that is eminently to accomplish concurrency is to build an object-oriented environment with support for messages passing between concurrent objects and minimize or eliminate the use of multiple threads of control with a single process [6]. In software development, Node.js^{7,8}, a cross-platform runtime environment for developing server-side web applications features technologies for designing RHM system.

The main contributions of this work are as follows:

- Design tasks that make sense to construct concurrent system for RHM application operates on the healthcare hub server located at the village hall. The design of components that constitute the healthcare hub server are also described.
- To investigate the performance of healthcare hub monitoring server that includes backend server, web programming framework and database able to support a large and increasing number of concurrent patients' requests. The performance metrics are throughput, response time and error rate to compare server-side JavaScript implementation using Google's V8 JavaScript engine and Rhino.

The rest of this paper is organized as follows. First, we discuss background of the technology related to the study in Section II. We elaborate a general system design model for building healthcare hub server in Section III. In Section IV, we brief the simulation, experimentation and analysis of results that state the performance of server-side JavaScript to handle concurrency connection in remote healthcare monitoring system. A conclusion in Section V sums up our performance of Node.js and the advantages should bring when deploying applications which require high-performance, asynchronous, event-driven network server with lowly resource requirements.

2. Background of technology related to the study

When it comes to server-side JavaScript programming, developers have choices between Rhino JavaScript engine and V8 based solutions like Node.js, SilkJS, and others⁹. Rhino⁹ is a Java-based JavaScript interpreter that gives JavaScript programs access to the entire Java API^{9,10}. The Node.js is a platform for building event-driven networking programs. It is a version of Google's V8 JavaScript. It runs single thread that makes it to serve many clients concurrently. As shown in the Fig. 2., the event loop queues both new requests and blocking I/O requests. The single-thread executes an event loop by setting up a simple mapping of all requests. The event loop gradually dequeues request from the queue, then processing the request, finally taking the next request or waiting for new requests submitted^{8,9}.

Using Node.js allows to write scalable network applications that inherit concurrency in their design. This is accomplished by event-loop. To avoid blocking, all main Application Programming Interface (API)-calls that involve Input/output are asynchronous. Thus, instead of waiting the function to return, the event-loop can execute the next event in the queue, and when the API-call is finished it calls a callback function specified when the call was made. There is an infinite loop running in a single process that continually looks at what event occurred and what callbacks need to be executed. It deals with queuing all events automatically, and it keeps calling the appropriate callbacks as fast as it can. As a developer using Node.js, you do not really need to know this, though you just need to know that your callbacks will be called when events occur as quickly as possible the task completes⁸. See Fig. 2.

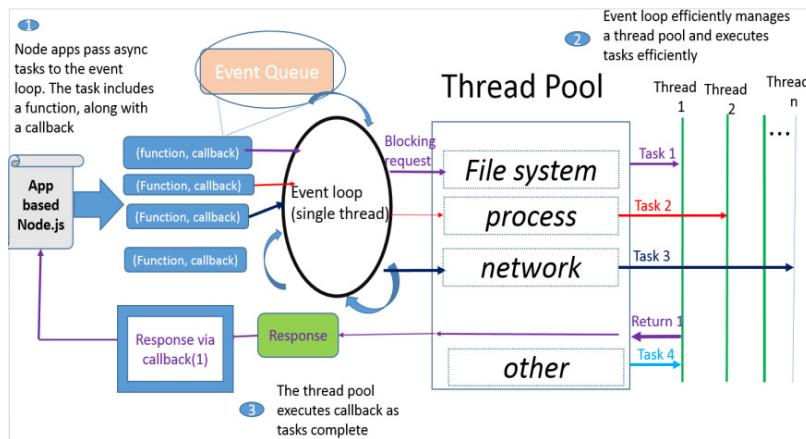


Fig.2. Node.js is based internally around the concept of an event loop. There is an infinite loop running in a single process that continually looks at what event occurred and what callbacks need to be executed. It deals with queuing all events automatically, and it keeps calling the appropriate callbacks as fast as it can [8].

3. Design of concurrency Remote Healthcare Monitoring on healthcare hub server

Taken by concurrent components, neither blocks of software nor data structures nor procedures make very natural models for concurrent components but objects seem like a very natural way to design a concurrency software⁶.

The Fig.3. illustrates objects as concurrent components for building RHM application on healthcare hub server. Consider an object model for our RHM system located at the village hall. A patient's status data object allows collection of patients 'data, then compares against a range of parameters set up by the doctor (physicians) object. Based on the results, the object model for our RHM send a message to series of actions to run object which executes a series of actions previously agreed upon by the patient and the doctor. Here execute actions by notifying the patient and updating patient's history data on doctor's (or physicians) tablet enable them to better assess the condition of the patients and reach more accurate diagnosis.

In order to support asynchronous communication, a current running instance of Node.js listens to all new incoming tasks. The running instance features event loop on which is provided with a message queue. The running instance refers to active object. The passive objects corresponds to the different tasks being run against the Node.js applications such as Range of parameters set up by the doctor, patient's data (Fig.3.). The passive objects include a function and a callback that corresponds to the response to be executed by the active object (thread pool) as soon the task encapsulates in the passive object is completed. The callback holds the state information of the passive objects that performed non-blocking and long tasks.

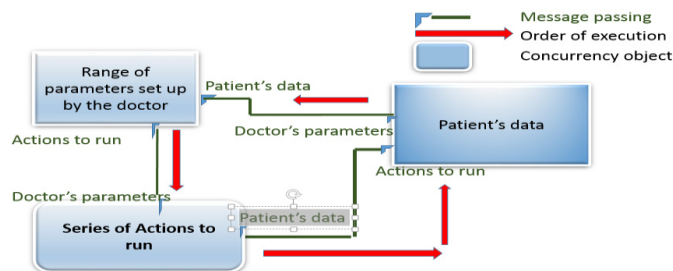


Fig. 3. Design of concurrent objects for building RHM application on healthcare hub server.

4. Building healthcare hub server using Node.js based and experimentation results

The test plan use JMeter¹¹ to capture throughput, response time results of Node.js's single-thread event-loop against the traditional application based java on the server-side JavaScript interpreter using the Apache sling. The capacity and performance testing is required to show that a healthcare hub monitoring server which consists of backend server and database layers can run with acceptable responsiveness when a large number of concurrent users can access these backend server-side and database simultaneously.

For each of the Client-healthcare hub Server Architecture, 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 Instant Heart Rate¹². It uses smartphone built-in camera to track color changes on the fingertip that are directly linked to your pulse. This android 's application is based on the mobile client server computing that has a module of uploading the accurate rate data stored on a temporary data store like SQLite¹³ to the remote healthcare monitoring application server. The test environment for the first Client- healthcare hub server Architecture consists of logic tier that determines how data can be created , displayed, stored ,changed and data tier as shown on the Table 1.

Table 1 implantation's components for healthcare hub server

Open platform under test for server side application	Logic tier	Data tier
Node.js	Express.js ¹⁴	MongoDB ¹⁴
Apache Sling ¹⁵	Java based server javaScript /Java Content Repository	MongoDB

The first tests against Node.js and the Apache sling simultaneously from 200 users up to 2000 users, 50 times. For each test, the "ramp-up" period is zero (0) which mean that all the users start sending the http request simultaneously.

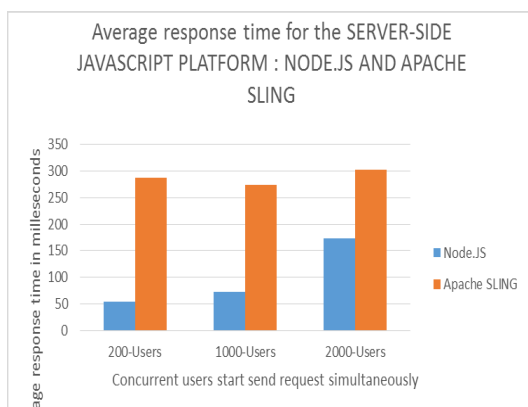


Fig. 4. Measurement of average response time

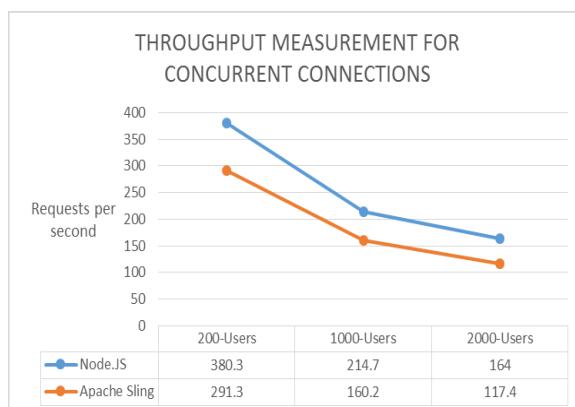


Fig. 5. The measurement of throughput for Node.js against Apache sling

What we can see on Fig.4. is 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-sling 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. The Fig. 5. shows that the throughput deteriorates for the two models as the number of users increases. The results show that for Node.js associated to MongoDB, the

throughput was 380 requests per second at 200 concurrent users, and 164 requests per second at 2000 concurrent users. The throughput of Node.js based event-driven approach using V8 runtime engine have performed better than Apache-sling which use rhino as server-side javaScript based Java. With Apache-sling-MongoDB configuration, the throughput increases up to 1000 concurrent users, then deteriorates as the number of concurrent users increases. This leads to formulate that the number of concurrent users carried out by an Apache-sling at server-side for building web services is not relatively constant. Therefore, Node.JS is roughly 40% faster, for example 164 request per second against 117 requests per seconds for 2000 users that corresponds to one hundred thousand (100000) concurrent requests.

5. Conclusion and future works

The Client-healthcare hub server constitutes by Node.JS server javaScript side and MongoDB is 40% faster that the Java based server side JavaScript solution using Apache sling with MongoDB database for implementing mobile client server computing related remote healthcare monitoring system. 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. For future work, we will implement the RHM system using the node.js that collects patients' data in a real time, controlling treatment recommendations and intervention based on the analysis of collected data.

Acknowledgements

This research was supported by the Brain Busan 21 Project in 2016 and Nurimaru R&BD project (Busan IT Industry Promotion Agency) in 2016

References

1. Yvette EG, Ha Jin H, Haeng Kon K. Internet of Things (IoT) Framework for u-healthcare System. *International Journal of Smart Home* 2015; Vol.9, No.11; p.323-330.
2. Niewolny D, White paper: How the Internet of Things Is Revolutionizing Healthcare. *Freescale Semiconductor, Inc., Reg. U.S.*; 2013.p.1-8.
3. Kang J, Yoo S and Oh D. Development of a Portable Embedded Patient Monitoring System.*International Journal of Multimedia and Ubiquitous Engineering (IJMUE)*. 2013; Vol.8, No.6; p.141-150
4. Kalin M. Three Ways to web Server Concurrency. Linuxjournal web site.2012, p1-2, Available at <http://www.linuxjournal.com/content/three-ways-web-server-concurrency>(last access April 2016)
5. PetitIV R.G, Goma Hassan. Analyzing Behavior of Concurrent Software Designs for Embedded Systems. *Proceedings of the 10 th IEEE International Symposium on Object and Component-Oriented Real-time Distributed Computing*.2007; p1-9
6. ObjectTime Limited. Designing for Concurrency.Canada; 1998, p1-28
7. Tilkov S, Vinoski S. Node.js: Using JavaScript to Build High-Performance Network Programs. In: Internet Computing, IEEE. 2010;Vol.14.No.6; p.80-83.
8. Yuhao Z,Daniel R,Matthew H,Vijay,JR, Microarchitectural implications of event-driven server-side web applications.In: Proceedings of the 48th International Symposium on Microarchitecture;2015.p. 762-774.
9. David F.JavaScript: The Definitive Guide, sixth Edition. O'Reilly Media, Gravenstein North, Sabastopol. United States of America; 2011
10. Artyom S. Continous Delivery of Apache Sling Applications. In: Ecolde Polytechnique Federale de Lausanne. Thesis presented for the Degree of Master of Science.Switzerland; 2014. Available on line at <https://bdelacretaz.files.wordpress.com/2014/09/artyom-thesis.pdf>.
11. EmilyH,H., "Apache JMeter. A practical beginner's guide to automated testing and performance measurement for your websites,PACKT PUBLISHING,BIRMINGHAM-MUMBAI,2008, pp:1-138
12. Shyam P. Building a Heart Rate monitor and dashboard for android, available on line at <https://www.pubnub.com/blog/2015-09-17-tutorial-realtime-android-heart-rate-monitor-and-dashboard/>
13. Sunguk L. Creating and Using Databases for Android Applications. In: International Journal of Database Theory and Application. 2012, Vol.5, No.2;p.99-106.
14. Simon Holmes. Getting Mean With Mongo, Express, Angular and Node. MEAP Edition. Manning Early Access Program. Manning, 2015, p8-24.
- 15.Apache Sling Architecture. <https://sling.apache.org/documentation/the-sling-engine/architecture.html>