모바일 웹 어플리케이션을 구현하기 위한 Node.js 파일에 대한 조사

라이오넬[*] · 장종욱[*]
[*]동의대학교

An Investigation into the Applicability of Node.js as a Platform for implementing Mobile Web Apps.

Lionel Nkenyereye[*] · Jong-Wook Jang[*]
[*]Dong-Eui University
E-mail : lionelnk82@gmail.com, jwjang@deu.ac.kr

요 약

본 논문에서는 오직 모바일 클라우드 컴퓨팅만을 사용하여 스마트폰 기반의 모바일 앱에서 Node.js 파일이 비동기 차단, 비 차단, 이벤트 기반 프로그램 패러다임을 제시한다. 또한 데이터베이스로 잘 알려진 MongoDB를 사용하여 App 사용자에 의해 전송된 방대한 데이터들을 처리한다. Node.js는 프로그래머가 동시 접속 문제를 해결하는 데 필요한 도구를 제공하는 것을 목표로 하고 있다. 원격 사용자들이 드라이버 입력을 전달하고 외부 응용 프로그램에서 출력을 제공하는 응용 프로그램을 고려하고 있는데, 차량을 이용하여 실시간으로 데이터를 분석 할 수 있는 스마트 폰 인터페이스 방식으로 응용 프로그램을 구현하여 제안 된 구조의 효과를 보여주고자 한다.

ABSTRACT

In this paper, we propose an architecture that affords mobile app based on nomadic smartphone using not only mobile cloud computing- architecture but also a dedicated web platform called Node.js built-in with the asynchronous, Nonblocking , Event-Driven programming paradigm. Furthermore, the design of the proposed architecture takes document oriented database known as MongoDB to deal with the large amount of data transmit by users of mobile web access application. The Node.js aims to give the programmers the tools needed to solves the large number of concurrent connections problem. We demonstrate the effectiveness of the proposed architecture by implementing an android application responsible of real time analysis by using a vehicle to applications smart phones interface approach that considers the smartphones to acts as a remote users which passes driver inputs and delivers output from external applications.

키워드

Client-Server Architecture, event-driven programming, Mobile Web App, Node.js, performance

Ⅰ. Introduction

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

one of the main challenge associated with building mobile application 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 concurent 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

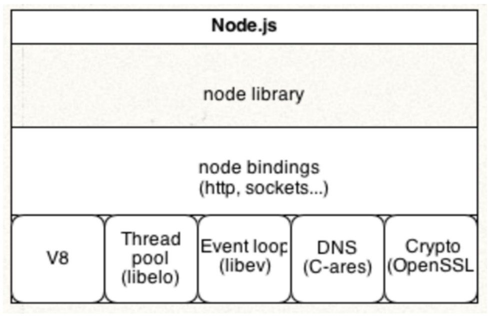


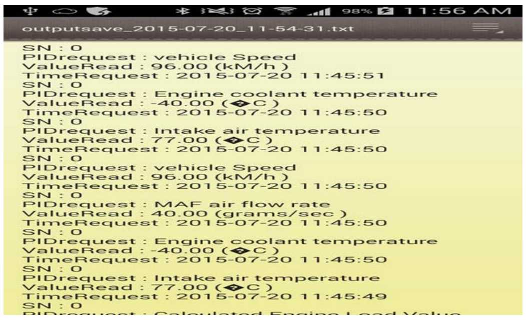
Figure 1. Internal structure of Node.js. the single thread handles all incoming requests[3].

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



```
outputsave_2015-07-20_11-54-31.txt
SN : 0
PIDrequest : vehicle Speed
ValueRead : 96.00 (kM/h )
TimeRequest : 2015-07-20 11:45:51
SN : 0
PIDrequest : Engine coolant temperature
ValueRead : -40.00 (�C )
TimeRequest : 2015-07-20 11:45:50
SN : 0
PIDrequest : Intake air temperature
ValueRead : 77.00 (�C )
TimeRequest : 2015-07-20 11:45:50
SN : 0
PIDrequest : vehicle Speed
ValueRead : 96.00 (kM/h )
TimeRequest : 2015-07-20 11:45:50
SN : 0
PIDrequest : MAF air flow rate
ValueRead : 40.00 (grams/sec )
TimeRequest : 2015-07-20 11:45:50
SN : 0
PIDrequest : Engine coolant temperature
ValueRead : -40.00 (�C )
TimeRequest : 2015-07-20 11:45:50
SN : 0
PIDrequest : Intake air temperature
ValueRead : 77.00 (�C )
TimeRequest : 2015-07-20 11:45:49
SN : 0
```

Figure 2. Saving vehicle's performance from web server to mobile device

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

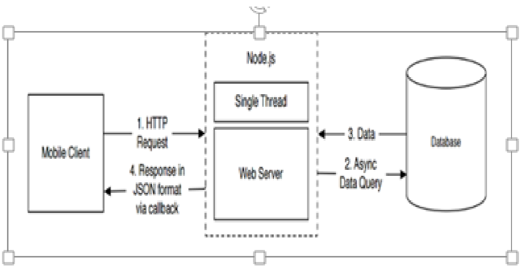
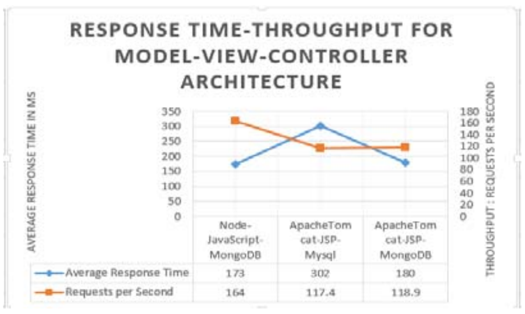


Figure 3. Design on how request from Mobile based application is submitted to Node.js.

Table 1 Scenarios cases for experimentation

#scenario	*Concurrent users*	*Ramp-up period(second)*	*Loop(times to run the similar sample)*
Scenario 1	200	0	50
Scenario 2	1000	0	50
Scenario 3	2000	0	50



	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 client-server Architecture. The measurement metrics are the throughput and response time

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

감사의 글

이 논문은 2016년도 한국연구재단의 지역혁신창의 인력양성사업과 Brain Busan 21 협력 기술개발사업의 지원을 받아 수행된 연구임을 밝힙니다.(NRF-2015H1C1A1035898, C0249807)

References

[1]Radek,V.,Roman,J., “Performance of Hybrid Mobile Application UI Frameworks”, Applied Mathematics, Computational Science and Engineering, pp: 293-306, 2013.

[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]Benjamin,S.,S., Maude,.L., “An Inside Look at the Architectural of NodeJS”,available on line http://mcgill-csus.github.io/student_projects/Submission2.pdf, last access, January, 2016

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

[5]Scantool, ECUsim 2000 OBD-II ECU simulator, https://www.scantool.net/ecusim-2000.html.

[6]Emily H,H., “Apache JMeter. A practical beginner's guide to automated testing and performance measurement for your websites, PACKTPUBLISHING,BIRMINGHAM-MUMBAI, pp:1-138,2008

[7]Brian,K., “CS764 Project Report: Adventures in Moodle Performance Analysis”, available on line at http://pages.cs.wisc.edu/~bpkroth/cs764/bpkroth_cs764_project_report.pdf, pp:1-28,last access, March 2016

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

[9]Florian,H.,Rene,P., “Performance optimization for querying social network data”,Workshop Proceedings of the EDBT/ICDT,pp:232-239,2014.