IoT 게이트웨이 기반 지능형 건물의 이벤트 중심 아키텍쳐 설계

라이오넬* · 장종욱*
*동의대학교

Design of IoT Gateway based Event-Driven Architecture for Intelligent Buildings.

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

요 약

모바일 기기는 사물 인터넷으로 성장하여 지능형 건물과 관련된 많은 IoT 응용 프로그램으로 연계 된다. 예를 들어 주택 자동화 제어 시스템은 스마트 폰으로 제어 명령을 보냄으로써, 홈 서버에 액세스를 하는 클라이언트 구조의 웹 어플리케이션을 요구한다. 홈 서버는 광 통신 시스템으로 명령어를 수신 받고 컨트롤 한다. 게이트웨이 기반 REST 기술은 클라이언트에서 요청하는 명령어를 처리 및 증명해야 한다. 이러한 이유는 클라이언트 요청에 의해 다수의 게이트웨이 증가로 인한 인터넷이 지연 되기 때문이다.
본 논문에서는 동시성 이벤트를 처리하기 위한 IoT 게이트웨이 시스템 설계를 하고자 한다. 본 시스템을 통하여 동시성 최고의 다중 추상화 레벨을 확인 할 수 있다. 동시성을 확인하는 방법은 개체 간의 데이터 통신을 지원하는 객체 지향 시스템을 구축하는 것이다. 또한 IoT 게이트웨이 기반으로 양방향 통신 방법 중 한쪽 통신 방향 프로토콜에 Node.js를 사용하여 이벤트 중심, 지능형 건물의 설계를 위한 아키텍쳐의 성능을 XMPP라는 미들웨어를 사용하여 확인하고자 한다. Node.js는 지능형 건물 제어 장치가 중앙 집중화 형식의 허브를 통하여 통신이 될 수 있도록 하는 역할을 가지고 있다. Node.js는 스레드 기반의 접근 방식이 특징이며, 기존의 시스템보다 40% 이상 빠르다. Node.js를 서버 측에서 사용하기 위해 다수의 클라이언트 들로부터 요청을 한다. 따라서, IoT 환경에서 지능형 건축물의 작업 수행 시간을 감소 시킨다.

ABSTRACT

The growth of mobile devices in Internet of Things (IoT) leads to a number of intelligent buildings 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 gateway based RESTful technology responsible for handling clients' requests attests an internet latency in case a large number of clients' requests submit toward the gateway increases. In this paper, we propose the design tasks of the IoT gateway for handling concurrency events. 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. We also investigate the performance of event-driven architecture for building IoT gateway using node.js on one side and communication protocol based message-oriented middleware known as XMPP to handle communications of intelligent building control devices connected to the gateway through a centralized hub. 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 intelligent building IoT environment.

키워드
IoT Gateway, Event-driven Architecture, Node.js, Intelligent buildings system, XMPP, Internet of Things.

Ⅰ. Introduction

The intelligent buildings refers to a range of Internet of Things (IoT) applications such as automatic energy metering, home automation and wireless monitoring. Traditionally, the intelligent buildings system use gateway based client server architecture such as web-based mobile application or web service to handle commands to control devices in IoT environment. As the number of clients' requests increases in simultaneously manner, the Representation State Transfer (REST) architecture to handle those requests encounters limitations for performing them successfully, and therefore, slows down predefined actions that take automatically without the help of humans.

This paper focuses on designing a prototype architecture for a gateway and its components of intelligent buildings for IoT applications. The model that we are looking for should have a gateway component with the ability of processing several requests from multiple smart devices simultaneously without blocking and long execution requests. These features are essential for reducing communication costs. In server-side asynchronous event-driven programming [2], the user requests are treated as application events and inserted into an event queue. This event-driven model has been widely adopted in building scalable web applications, mainly through the Node.js [1] framework. The real-time communication allows the implementation of eXtensible Messaging and Presence Protocol (XMPP) to support the IoT device management framework [3]. The implementation of XMPP provides near-real-time communication between multiple devices over multiple networks

Ⅱ. Prototype Architecture for Intelligent Buildings related IoT applications with its gateway

The design prototype shown in Fig. 1 consists of database and virtual machines as resources to carry out several functions from the intelligent building controls units. The intelligent building controls units are responsible of sending the measurement of sensors installed on the microcontroller boards. The components on the cloud store readings and status of these intelligent building control units (IBCU) periodically for further analysis. The components on the cloud are also responsible of sending control and monitoring commands to the sensors connected to these IBCUs. The IBCUs are the IoT devices nodes in charge of manage the data to be sent to the management components services on the

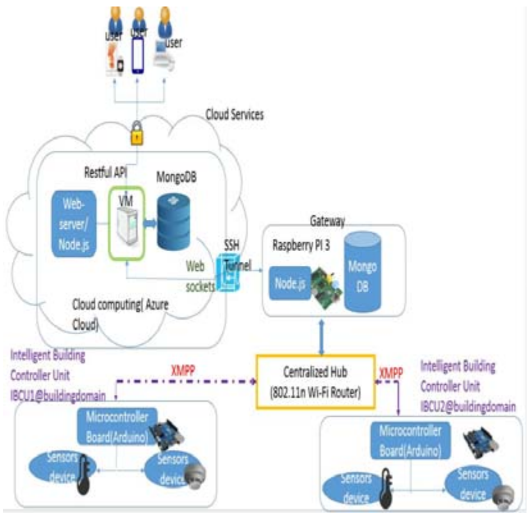

user
user
user
Cloud Services
Restful API
MongoDB
Web-server/Node.js
VM
SSH
Tunnel
Web sockets
Cloud computing(Azure Cloud)
Gateway
Raspberry PI 3
Node.js
Mongo DB
Intelligent Building Controller Unit IBCU1@buildingdomain
XMPP
Centralized Hub (802.11n Wi-Fi Router)
XMPP
Intelligent Building Controller Unit IBCU2@buildingdomain
Microcontroller Board(Arduino)
Sensors device
Sensors device
Microcontroller Board(Arduino)
Sensors device
Sensors device

Figure 1 A prototype architecture for Intelligent Buildings (energy metering, home automation, wireless monitoring) related IoT environment.

cloud. Every IBCUs connected to the Wi-Fi network uses a unique IP address that can be used to reach that IBCU. At the IBCUs level, basic instructions are implemented in order to pre-process information on the main gateway. The client application accessible anywhere through either wearable device or mobile device discovers all the connected IBCUs and enables the user to start interacting with them. Each IBCUs broadcasts itself over the network so that the cloud services can discover these IBCUs for reporting to the

user.
The Raspberry Pi 3 is the main gateway, whose is to interconnect the IBCUs to the network providers' infrastructure of the system. This gateway runs a node.js in order to process a large number of concurrent connections from the end users using the intelligent building applications. The web socket ensures the communication between the cloud services and the raspberry Pi through an encrypted secure shell (SSH) tunnel. This gateway is also in charge of sending instruction set which is in the format of commands that can be communicated for performing automatic configuration when a new IBCU is connected to the intelligent building related IoT system. The instructions are implemented in JavaScript and are communicated to the microcontroller board (Arduino) devices to request sensors readings or to send commands to control the peripheral sensors.
Finally, the end users are able to interact with the whole intelligent building related IoT applications using a secured REST API, using mobile device and connected wearable devices. A smartphone application through cloud services that give access to the gateway made for the purpose of controlling all the IBCUs currently available on the network. This is done by the functionality of the XMPP protocol. Upon retrieving the IBCUs and make request over the network in the form of messages that are sent to the IBCUs through the centralized hub.

Ⅲ. Performance Investigation of IoT gateway implemented using server-side framework based Event-driven against no JavaScript server side

To measure the performance of different use cases the program Apache JMeter 2.712 was used [4]. The component under test was the main back end server. The simulation of client-server architecture is presented on the Fig. 2. 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 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

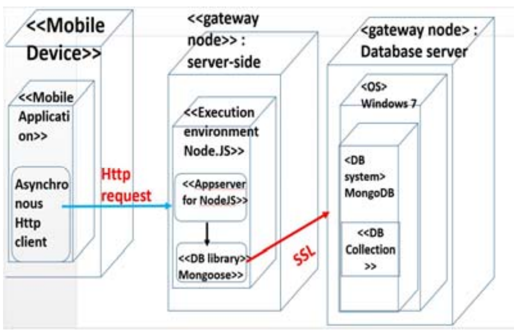
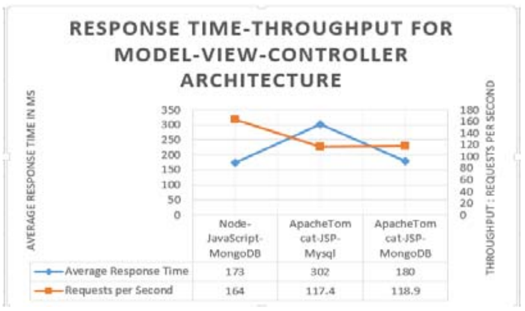


Figure 2 A deployment diagram of a Node.js system. It depicts Asynchronous Http client and communicate with the Node.js server application running on windows server.



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

Figure 3 The performance of the three model client-server Architecture. The measurement metrics are the throughput and response time

The Figure 3 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.

Ⅳ. Conclusions and Future Work

This paper presented a design a prototype architecture for intelligent building that attests the capability of performing a large number of users accessing the intelligent buildings system concurrently. The XMPP protocol allows multiple connected devices to provide real-time communications over multiple networks. With the use of Node.js, an increasing number of requests should not have a negative effect when the information collected from sensors embedded in multiple connected devices is used as rule engines that support the formulation of decision logics.

Acknowledgment

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

References

[1] 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*, (2015), 762-774.

[2] Dabek, N., Zeldovich, N., Kaashoek, F., Mazieres, D. and Morris, R., Event-driven programming for robust software. *Proceeding of SIGOPS. European Workshop*, (2002), 186-189.

[3] Sung-Chan, C., Jaeho, K., Jaeseok, Y. and El-Yeop, A., A Tutorial for Energy-efficient Communication for XMPP-based Internet of Things. Smart Computing Review, 306 (2013), 471-479.

[4]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