北京联合大学毕业设计（论文）外文原文及译文

题目：团队协同办公系统

专业：计算机科学与技术　　指导教师：孙悦

学院：信息学院　　学号：2013080332029

班级：计科1303B　　姓名：王庭旭

一、外文原文

ELSEVIER

CrossMark

Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 98 (2016) 382 – 387

Procedia Computer Science

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.

Peer-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;

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 cloud[1].

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 settings[2]. However, most of RHM technologies follow a general architecture that consists of four components[2]. 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 meet[3]. 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. Furthmore, the patient should not benefit any immediate medical attention.

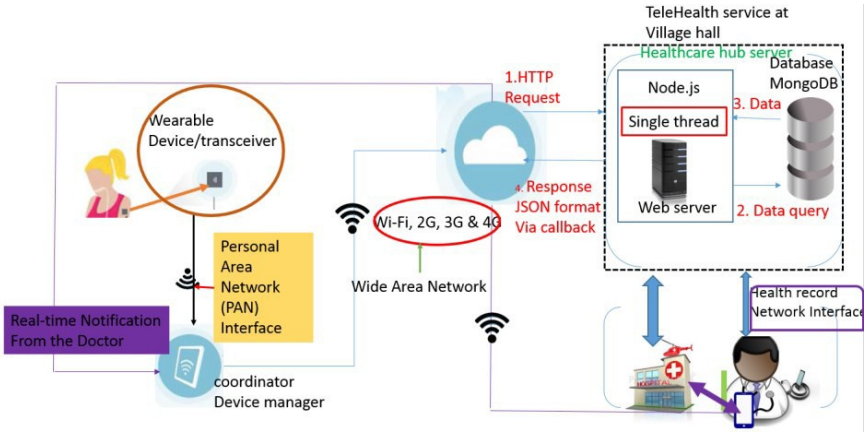


Fig. 1. Architecture of Remote healthcare monitoring system in an isolated location with no access to local medical care.

When we talk about concurrency, two approaches are proposed[4]. The first approach is to have multiple threads in a single process. The second approach is to have multiple single-threated processes. Using separate processes for concurrency also has an additional advantage. The advantage is to run the separates 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 sequences 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 include 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 abstraction[5]. First, the functional requirement of the RHM system must be well understood in term of its desired behavior. Next, possible function 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 others[9]. Rhino[9] 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 dequeuing 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 completes[8]. See Fig. 2.

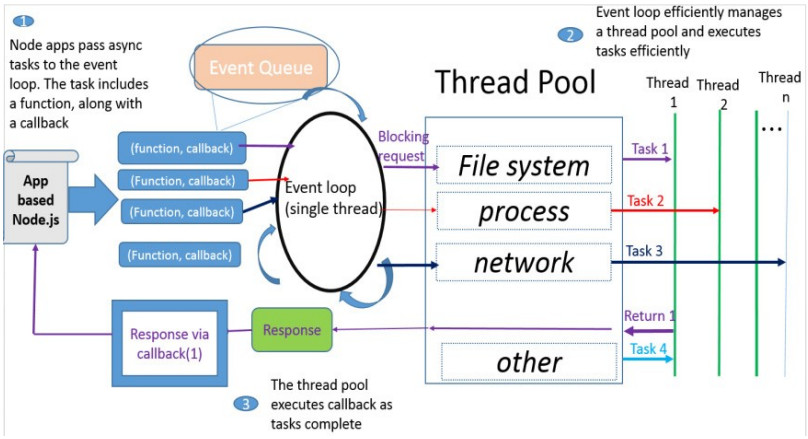


Fig.2. Node.js is based internally around the concept of an event loop. There is an infinite loop running in a single process that continually looks at what event occurred and what callbacks need to be executed. It deals with queuing all events automatically, and it keeps calling the appropriate callbacks 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 software[6].

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.

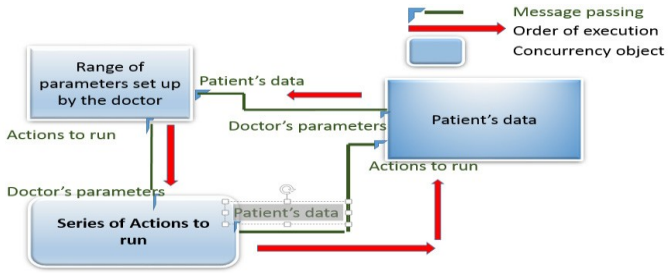


Fig. 3. Design of concurrent objects for building RHM application on healthcare hub server.

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

The test plan use JMeter[11] 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 minitoring 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 Rate[12]. 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 SQLite[13] to the remote healthcare monitorting 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.

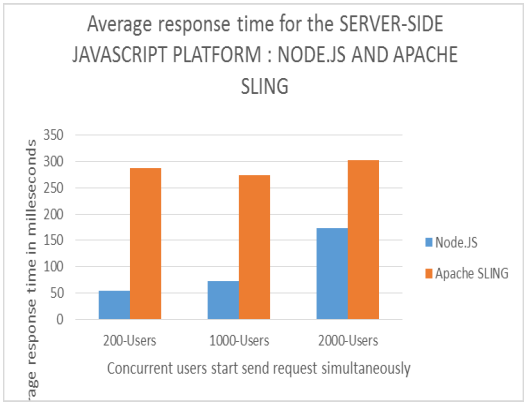


Fig. 4. Measurement of average response time

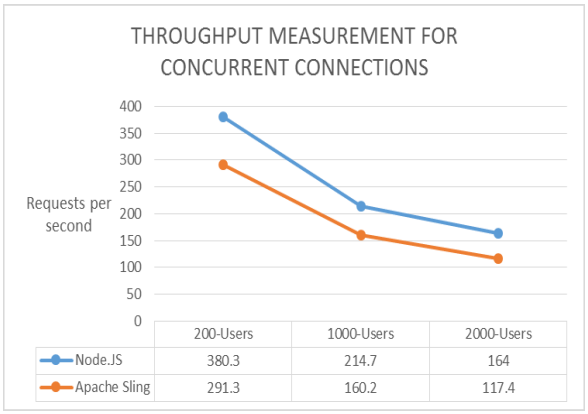


Fig. 5. The measurement of throughput for Node.js against Apache sling

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

二、译文

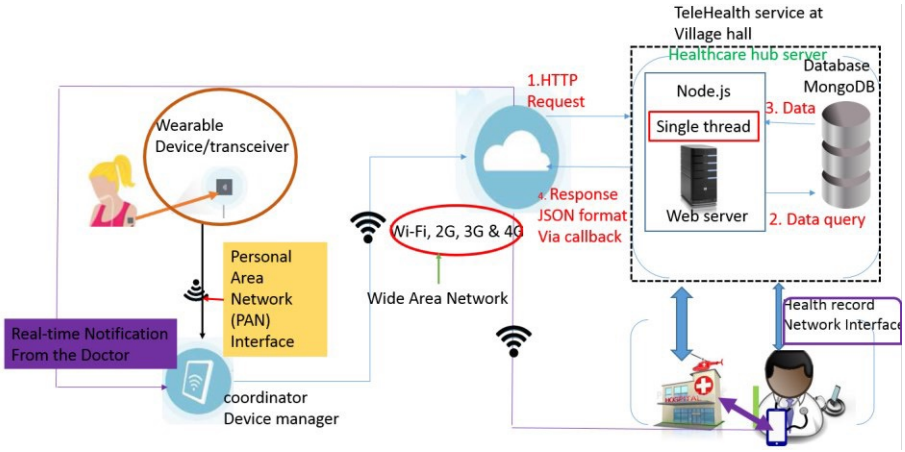
医疗中心服务器在远程医疗监控系统中的服务器端渲染 javascript 性能评估

1. 介绍

像物联网医疗系统这样的产品将设备连接起来组成一个巨大的系统的点子是基于物联网（IoT）。这个物联网医疗系统是利用连接在一起的设备所形成的网络利用无线连接以及因特网访问能力共享有用的信息到云端的中央处理器以及控制服务器。

远程医疗监测（RHM）是一种技术，用于监测居住在一个孤立的村庄的患者，无法获得对常规临床设置的有效健康监测。然而，大多数 RHM 技术遵循由四个组件组成的通用架构。第一组件是通过无线通信以测量生理参数的各种设备传感器。第二组件是在患者站点处的协调器移动设备（智能电话）上的本地数据存储，其在传感器和集中式数据存储库与健康监护集线器服务器之间进行接口。协调器设备配备有无线连接，例如 Wi-Fi 或 3G / 4G LTE 启用的互联网，以向医疗保健提供者发送收集的生理数据。第三个组件是存储从传感器，本地数据存储，诊断应用程序和医疗保健提供者发送的数据的集中式存储库。最后一个组件是诊断应用软件，其基于通过触发先前由患者和医生商定的一系列动作基于对收集的数据的分析来向患者发送实时警报。在本文中，我们使用术语“保健中心监视服务器”来引用第三和第四部分。图 1 显示的是 RHM 系统的架构。

RHM 系统的健康护理中心监视服务器旨在以并发方式处理来自患者的大量请求，以有效地处理治疗建议。并发性是在系统中同时发生的事情的趋势。并发性在 RHM 系统中特别重要，它们必须实时响应外部事件，并且通常有严格的最后期限来满足。随着患者请求的数量以同步的方式增加，位于村庄厅的保健中心服务器会遇到瓶颈。因此，该 RHM 应用不能检测患者健康的恶化，减慢患者和医生先前约定的预定义动作。作为后果，急诊部门访问次数增加，住院治疗和住院时间延长了医疗保健的交付成本。此外，患者不能接收实时警报，这是在医生的帮助下自动发生的一系列动作。这些警报提醒患者采取一些预防措施。当医生无法从医疗中枢服务器接收新数据和患者的历史时，这种情况可能导致患者的死亡。此外，患者不应受益于任何立即的医疗护理。



图一. 远程医疗监测系统架构在孤立的位置，无法获得当地医疗。

当我们谈论并发时，提出了两种方法。第一种方法是在单个进程中拥有多个线程。第二种方法是具有多个单次处理过程。使用单独的并发进程也有一个额外的优点。优点是在通过网络连接的不同机器上运行分离进程。虽然这增加了通信成本。另一种方法是在单个进程中运行多个线程。每个线程独立于其他线程运行，并且每个线程可以运行不同的指令序列。多线程方法的缺点是单个进程中的所有线程共享相同的资源。在共享资源失败的情况下，在单个进程中包括的活动线程也被破坏。

在本文中，我们提出了 RHM 应用程序的设计任务，用于处理并发任务。在设计任务的过程中，通过采用多级抽象来最好地理解并发。首先，RHM 系统的功能需求必须被充分地理解以获得期望的行为 。接下来，应检查可能的并发功能。这最好使用线程的抽象。明显地实现并发的方式是构建面向对象的环境，支持消息在并发对象之间传递，并通过单个进程最小化或消除对多线程控制的使用。在软件开发中，用于开发服务器端 Web 应用程序的跨平台运行时环境 Node. js 具有用于设计 RHM 系统的技术。

这项工作的主要贡献如下：

- 为 RHM 应用程序构建并行系统有意义的设计任务在位于村庄大厅的医疗中心服务器上运行。还描述了构成保健中心服务器的组件的设计。
- 为了研究包括后端服务器，Web 编程框架和数据库的医疗中心监控服务器的性能，能够支持大量并且不断增加的并发患者请求。性能指标是吞吐量，响应时间和错误率，以便使用 Google 的 V8 javaScript 引擎和 Rhino 来比较服务器端的 javaScript 实现。

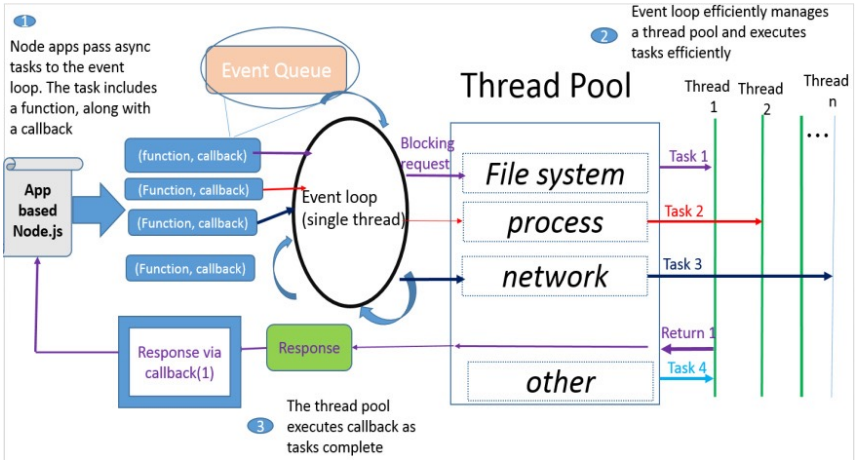
本文的其余部分安排如下。首先，我们讨论第二部分研究相关技术的背景。我们在第三部分中阐述了建立医疗中心服务器的一般系统设计模型。在第四部分，我们简要介绍模拟，实验和分析的结果，说明服务器端 javaScript 处理并发连接的远程医疗监护系统的性能。第五节中的结论总结了

我们 Node.js 的性能，以及在部署需要高性能，异步，事件驱动的网络服务器且需求低的应用程序时应具有的优势。

2. 与研究相关的技术背景

当涉及到服务器端 JavaScript 编程时，开发人员可以在 Rhino javaScript 引擎和基于 V8 的解决方案（如 Node.js，SilkJS 和其他）之间进行选择。 Rhino 是一个基于 java 的 JavaScript 解释器，它允许 JavaScript 程序访问整个 Java API。 Node.js 是构建事件驱动的网络程序的平台。这是一个谷歌的 V8 JavaScript 引擎。它运行单线程，使其同时服务于多个客户端。如图 1 所示。事件循环对新请求和阻塞 I / O 请求进行排队。单线程通过设置所有请求的简单映射来执行事件循环。事件循环逐渐从队列中出队请求，然后处理请求，最后获取下一个请求或等待新的请求提交。

使用 Node.js 允许编写在其设计中继承并发性的可扩展网络应用程序。这是由事件循环完成的。为了避免阻塞，涉及输入/输出的所有主要应用程序编程接口（API） - 调用是异步的。因此，事件循环可以执行队列中的下一个事件，而不是等待函数返回，而当 API 调用完成时，它调用在调用时指定的回调函数。有一个无限循环在单个进程中运行，持续查看发生了什么事件和需要执行什么回调。它处理自动排队所有事件，并且它尽可能快地调用适当的回调。作为使用 Node.js 的开发人员，你并不需要知道这一点，尽管你只需要知道当事件发生时尽快调用回调函数。参见图 2。



图二. Node.js内部围绕着事件循环的概念。有一个无限循环在单个进程中运行，持续查看发生了什么事件和需要执行什么回调。它自动处理排队的所有事件， 保持调用适当的回调尽可能快。

3. 在医疗中心服务器上设计并发远程医疗监控

由并发组件采用，软件和数据结构的块或程序都不能为并发组件创建非常自然的模型，但对象似乎是设计并发软件的一种非常自然的方式。

图 3 将对象图示为在医疗中心服务器上构建 RHM 应用程序的并发组件。考虑我们位于村庄大厅的 RHM 系统的对象模型。患者的状态数据对象允许收集患者的数据，然后与由医生（医生）对象设置的一系列参数进行比较。基于结果，我们的 RHM 的对象模型发送消息到一系列动作以运行对象，其执行先前由患者和医生商定的一系列动作。这里通过通知患者并更新医生（或医生）片剂的患者历史数据来执行动作，使得他们能够更好地评估患者的状况并且达到更准确的诊断。

为了支持异步通信，Node.js 的当前运行实例侦听所有新的传入任务。运行实例的特征在于其上提供有消息队列的事件循环。运行实例引用活动对象。被动对象对应于针对 Node.js 应用程序运行的不同任务，例如由医生设置的参数范围，患者数据（图 3）。被动对象包括与待被封装在被动对象中的任务完成时对应于由活动对象（线程池）执行的响应相对应的函数和回调。回调保存执行非阻塞和长任务的被动对象的状态信息。

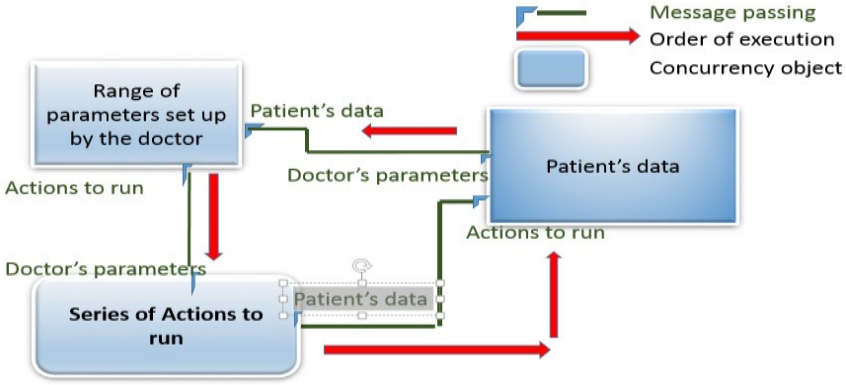


图 3. 在医疗中心服务器上设计用于 RHM 应用程序的并发对象。

4. 使用基于 Node.js 和实验结果构建医疗中心服务器

测试计划使用 JMeter11 捕获吞吐量，Node.Js 的单线程事件循环的响应时间结果与传统的基于应用程序的 java 在使用 Apache sling 的服务器端 JavaScript 解释器上的结果。需要进行容量和性能测试，以显示由后端服务器和数据库层组成的医疗中心监视服务器可以在大量并发用户可以同时访问这些后端服务器端和数据库时以可接受的响应速度运行。

对于每个客户端 - 保健中心服务器架构，目标不是测试 web 应用，而是以与从 web 浏览器提交 http 请求相同的方式监听从移动设备发送的 http 请求。被测应用程序是即时心率。它使用智能手机内置相机来跟踪指尖上与您的脉搏直接相关的颜色变化。这个 android 的应用程序是基于移动客户端服务器计算，其具有将存储在临时数据存储（如 SQLite）上的准确速率数据上传到远程医疗监护应用服务器的模块。第一个客户端 - 保健中心服务器架构的测试环境包括确定如何创建，显示，存储，更改数据和数据层的逻辑层 。

第一个测试针对 Node. js 和 Apache 吊带同时从 200 个用户到 2000 个用户，50 次。对于每个测试，“加速”时段是零，这意味着所有用户同时开始发送 http 请求。

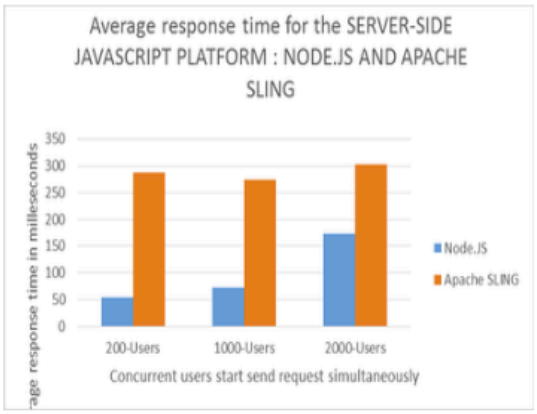


Fig. 4. Measurement of average response time

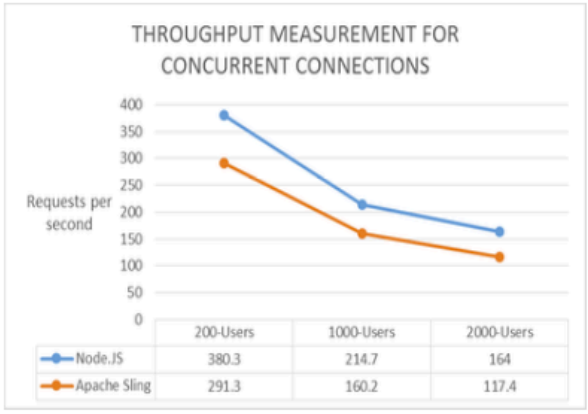


Fig. 5. The measurement of throughput for Node.js against Apache sling

我们可以看到图 4。是响应时间随着并发请求数的增加而降低。例如，Node. JS-MongoDB 在 200 个并发用户的平均响应时间为 54ms，平均为 2000 个并发请求的响应时间为 173ms。我们可以看到，对于服务器端的 Apache-sling，平均响应时间与并发请求数几乎呈线性相关。这意味着并发用户数千倍的增长开始响应时间增加 100 倍。 。示出随着用户数量的增加，两个模型的吞吐量恶化。结果表明，对于与 MongoDB 相关的 Node. js，

吞吐量为每秒 200 个并发用户的 380 个请求，以及 2000 个并发用户每秒 164 个请求。使用 V8 运行时引擎的基于 Node. js 的事件驱动方法的吞吐量比使用 rhino 作为基于服务器端基于 JavaScript 的 Java 的 Apache-sling 性能更好。使用 Apache-sling-MongoDB 配置，吞吐量可增加最多 1000 个并发用户，然后随着并发用户数量的增加而恶化。这导致了在服务器端用于构建 web 服务的 Apache-sling 所执行的并发用户的数量不是相对恒定的。因此，Node. JS 大约快 40%，例如对于对应于十万（100000）并发请求的 2000 个用户，每秒 164 个请求/秒，对于 117 个请求。

5. 结论和未来的工作

客户端 - 保健中心服务器由 Node. JS 服务器构成，JavaScript 端和 MongoDB 的速度比基于 Java 的服务器端 JavaScript 解决方案快 40%，使用 Apache 吊带与 MongoDB 数据库实现移动客户端服务器计算相关的远程医疗监护系统。在本文中，单线程事件循环 Node. js 和多线程方法之间的差异并发模型有所不同。为了测试 Node. js 更高的并发级别，它应该超越多线程，其他问题，如增

加请求的数量。对于未来的工作，我们将使用 node.js 实施 RHM 系统，该实时收集患者的数据，基于收集的数据的分析来控制治疗建议和干预。