MobisenseCar: A Mobile Crowd-Based Architecture for Data Acquisition and Processing in Vehicle-Based Sensing

Lionel Nkenyereye
Department of Computer Engineering
Dong-Eui University
Busan, Republic of Korea

Jong Wook Jang
Department of Computer Engineering
Dong-Eui University
Busan, Republic of Korea

***Abstract*—The use of wireless technology via smartphone allows designing smartphone applications based on OBD-II for increasing environment sensing. However, uploading of vehicle's diagnostics data via car driver's tethered smart phone attests a long Internet latency when a large number of concurrent users use the remote mobile crowdsensing server application simultaneously, which increases the communication cost. The large volume of data would also challenge the traditional data processing framework. This paper studies design functionalities of mobile crowdsensing architecture applied to vehicle-based sensing for handling a huge amount of sensor data collected by those vehicle-based sensors equipped with a smart device connected to the OBD-II interface. The proposed MobiSenseCar uses Node.js, a web server architecture based on single-thread event loop approach and Apache Hive platform responsible for analyzing vehicle's engine data. The Node.js is 40% faster than the traditional web server side features thread-based approach. Experiment results show that MapReduce algorithm is highly scalable and optimized for distributed computing. With this mobile crowdsensing architecture it was possible to monitor information of car's diagnostic system condition in real time, improving driving ability and protect the environment by reducing vehicle emissions.**

***Keywords*—Mobile crowdsensing; data processing; web services; hadoop; hiveQL; OBD-II**

I. Introduction

Mobile crowdsensing refers to a new paradigm that allows a certain number of individuals to collectively share data and extract information so as to measure and map phenomena of common interests [1]. Improvements in terms of smart device capabilities and communication technologies allow crowdsensing solutions to emerge as significant strategies to revolutionize environment sensing. If vehicular mobility is adopted, the sensing capabilities can be further increased by connecting smart devices to vehicles using the On Board Diagnostic interface (OBD-II) or provided directly by smart vehicles through vehicle-to-vehicle or vehicle-to-infrastructure communications [2].

Remote On-line Vehicle Diagnostics (ROVD) is such a telematics service that provides opportunities to constantly monitor the vehicle diagnostics system remotely [3]. ROVD integrates the capability of computing unit to identify a fault or a possibility of a fault in an automobile and transmit vehicle trouble codes to a remote central processing center or public cloud computing [3]. Thus, ROVD replaces wireless technology over short distances or cable between on-board vehicle's connector and vehicle's monitoring system by mobile network based data [4]. These vehicle monitoring systems move into a remote data center owned by car manufacturers for instance. ROVD also adds the event-driven intelligence in which the vehicle determines when it is necessary to inform changes in the vehicle's status [4].

The number of mobile crowdsensing vehicle's diagnostics applications (MCVDAs) has increased in the automotive market [3]. These MCVDAs ensure that the messages received from the vehicle's engine reach the smartphone through the Bluetooth wireless interface and the smartphone's wireless communication modules transmit the received packets to the remote data center. Thus, these applications may play an important role in reducing the number of accidents and allowing the automotive industry to use data generated from an automotive electronic system to make new models of cars in the future. Another advantage is that it allows Original Equipment Manufacturer (OEM) or its service partner to inform the driver whether he can continue his journey safely or whether he requires assistance from a near repair shop. This helps prevent vehicle breakdown by detecting the vehicle problems at an early stage.

The transmission of vehicle's diagnostics data via car driver's tethered smart phone increases the cost in communication by looking at the amount of generated data in order of a few gigabytes per hour per vehicle [5]. As a consequence, the vehicle commercial company or car manufacturers and automobile ecosystem would have to support the lifetime cost of diagnostics data. It is not reasonable for the car manufactures or OEM to support communication cost when they cannot control the usage of the mobile application delivered to the car owner. The data transmission may not only increase cost communication but also internet latency when the architecture of the remote web server may not process concurrent requests from a large number of drivers uploading their vehicle's diagnostics data simultaneously. For example, the driver would interrupt the uploading of a vehicle's engine diagnostics data when he (she) uses other applications installed on his (her) smart phone due to the unexpected communication latency. In this case, the OEM or its service partner would not be able to start a diagnosis session remotely, and therefore, leave the driver with possibility of not receiving any assistance. Furthermore, the enormous quantities of vehicle's diagnosis data generated by the MCVDA may challenge its analysis making it difficult for the car manufacturers' technicians to figure out the cause of vehicle breakdown quickly, for instance.

The connectivity inside the vehicle may be established. However, due to the mobility of vehicles and the lack of a reliable connection, a real-time uploading would be unsuccessful. For this, the pushing technique from a temporary storage should be used on the mobile application to the remote server for realizing data replication after a certain distance of driving.

This study proposed a mobile crowdsensing architecture for designing MCVDA that relies on the tethered connectivity model, component-based mobile applications, Client-Server Architecture and data processing framework for processing volume of both structured and unstructured data. Therefore, the tethered connectivity model carries the smart phone inside the vehicle. The smart phone is used as a modem via wired, Bluetooth or Wi-Fi as communications technologies. The component-based mobile application and Client-Server Architecture feature have the ability to process a large number of requests from multiple drivers simultaneously. The proposed Client-Server Architecture prevents blocking and long execution requests that may increase communication cost. To solve the limitation of blocking running requests, the Client-Server architecture handles them using a single-thread event loop web server-side called Node.js [6], [7]. The data processing model is implemented using Big data technology which provides infrastructure that can manage and process vehicle-based sensing data, thus enabling to enhance safety and driving experience. The implantation of Big data includes Hadoop, an open source distributed platform for storing and processing data. This Hadoop includes the Hadoop Distributes File System (HDFS) which provides Application Program Interfaces (APIs) for MapReduce applications to read and write data in a parallel manner [8].

The main contribution of this work is as follows:

- Design components based mobile client server computing architecture is introduced for implementing the MCVDA application called MobiSenseCar which is based on Android. The design of the MobiSenseCar, a mobile application allows the managing of the issue of concurrent clients' requests and replication of vehicle's diagnostic data on a temporary SQLite database on a mobile application to solve the issue of unreliable connectivity.
- A novel analytical model is based on big data technology for monitoring vehicle's diagnostics data. In-vehicle's diagnostic data is stored in HDFS; analyzing jobs are executed by queries in HiveQL language, and the queries will be transferred into MapReduce progress.
- Investigate the performance of data acquisition platform which consists of a Client-Server Architecture that includes backend server, web programming framework and database that can 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.

The rest of this paper is organized as follows. First, the related work is discussed in Section II, then focus on the background of technology of event-driven approach, Node.js and Big Data technology in Section III. This section elaborates a general system design model for MobiSenseCar based on web server with Node.js and Hadoop. Next, the paper takes an isolated look at different stages of this model and gets to know their relevant components approaches to adopt in Section IV. In Section V, it discusses the results of the implementation of a MobiSenseCar mobile application based on Android that features different components of the proposed mobile client server computing architecture. In Section VI, it briefs the simulation, experimentation and analysis of results of the proposed mobile crowdsensing architecture. The conclusion in Section VII sums up the architecture for a diagnosis of the status of a vehicle's engine and the advantages of deploying applications based on tethered connectivity model for leveraging heterogeneous crowdsourced data from MCVDA.

II. Related Work

Currently there are few ad hoc solutions to smart phone based sensing vehicle monitoring systems. Therefore, prototype model includes on-board computer, wireless communication link, vehicle monitor server, and vehicle status browser. Below the paper proceeds by describing briefly the different smart phone based mobile crowdsensing architecture for monitoring the car as a sensing platform.

Prashanth et al. [9] made an architecture that leverages sensors besides GPS-accelerometer and microphone, in particular to glean rich information such as the quality of the road or the noisiness or traffic. They proposed algorithms to virtually reorient a disoriented accelerometer along a canonical set of axes and then use simple threshold-based heuristics to detect bumps and potholes and braking. Their study focus on the use of sensing components in the smart phone applied to proposed algorithms for detecting potholes. A similar application is PotHole [10] which can identify holes in streets using the crowdsourced vibration and position data collected from smartphone.

Recently, both Derick et al. [11], Eren et al. [12] and Mohamed et al. [13] propose a driving style recognition application using smartphone as a sensor platform. The evaluation in [11] proves that classical Dynamic Time Warping algorithm can accurately detect events with a very limited training set. In [12], safe or unsafe optimal path detection algorithm and Bayesian classification applied to vehicle data detect the driver behavior, and then increase safety while driving.

Jules et al. [14] present a formal model for accident detection that combines sensors and context data. They showed how smartphone sensors, network connections, and web services can be used to provide situational awareness to first responders. The contribution of their work provides empirical results demonstrating the efficacy of different approaches employed by smartphone accident detection systems to prevent false positives.

There is another interesting work which proposes driver behavior profiling using smartphone [15]. This work analyzed how smartphone sensors can be used for identifying maneuvers and propose a platform that is able to detect risky driving events independently from the mobile device and vehicle. In this work, the fuzzy logic event detection mechanism is implemented in an Android application. The authors state that the approach intends to use the DoIP protocol to perform vehicle diagnostics data exchange synchronously over a TCP connection over wireless communication and information infrastructures.

The design prototype proposed in the above studies has some drawbacks. The first drawback is that the design prototype proposed in [10], [11], [14], [15] does not take into consideration the impact of a large amount of generated data. How and where to preprocess and aggregate Controller Area Network (CAN) data is also an important question which was not addressed. Furthermore, there is no performance evaluation of Client-Server Architecture designed to handle a large number of requests submitted from participatory vehicle – based sensing platform. The second drawback is about the integration of the data processing framework for analyzing a huge amount of vehicle's diagnostics data generated by MCVDA that would enhance decision making.

At one of the range, there is rich commercial vehicle's telematics application as mentioned in the white paper published by Oracle. In that white paper, the Oracle for the Connected Vehicle highlights how their Octo Telematics solution architecture would turn data generated by vehicles into business, therefore enhancing the value of vehicle's diagnostics data [16]. Thus, the proposed Octo telematics solution relies on the embedded "clear box" of sensors used to collect on-board vehicle's data, and then uploads them to the central message system where a mechanism of validation meshes unto the insurance company's framework.

III. Background of Technology

The purpose of this section is to provide an introduction and background to areas immediately tied to this work. It describes the characteristics of some important fields that should be taken into consideration when conducting this kind of studies. First, it describes the characteristics of the connectivity models inside vehicles that might provide a feasible integration of solution based on them. Secondly, it provides the reader with a short introduction to server-side scripting to respond to network and concurrent requests. Finally, it describes the big data technology and its related framework to process large data sets of information.

A. Communication Solution Inside Vehicles

The wireless communication technologies built-in or brought in the vehicle will enable the automotive world to afford new applications such as navigation, billing facilities, and fleet management. The success of new in-car telematics applications and services can be realizable through the V2I (Vehicle-to Infrastructure) data exchange with network operators [17]. The connectivity inside the vehicle may be established by the network operator in three ways: embedded solution, tethered solution and integrated solution [17].

- The embedded solution: it includes both connectivity and intelligence that are built into the vehicle. For this kind of connectivity, the in-car telematics services proposed by the auto makers will force them to build TCU (Telematics Control Units) into the vehicle within a SIM (Subscriber Identity Module) card to connect to the network in order to make calls, and receive texts for instance.
- The tethered solution: The tethered connectivity model stands on the obligation of carrying the smart phone inside the vehicle. This smart phone is used as a modem via wires, Bluetooth or Wi-Fi.
- The integrated solution: integrated solution is considered as an integration of the smart phone application into the vehicle to enable the driver to access telematics or cloud-based applications and their features safely while driving.

The research work done on the growth of connectivity model inside the vehicle and published by the World Leading Knowledge Partners to the Automotive industry [17] states that tethered solutions will grow more and eventually peak in developed USA and Europe market by the end of the decade [17]. As embeddedness is considered to be the seamless and reliable solution for the future in-car telematics service, USA and Europe markets are interested in tethered solution for the only reason that tethered solutions are considered as short-term solutions to customers' unwillingness to pay a second communication cost for vehicle-related connectivity in the car on the top of their mobile phone communication bill [17].

B. Server-side Scripting Approach

The architecture of web server-side and its scripting approach must inherit features that allow responding to an increasing number of network requests from the end-users. Thread-based scripting approach has been used to implement web application to respond to the clients' requests. However, web server based on the thread-based approach might perform inefficiently as the number of incoming network requests increases. Therefore, many industries such as eBay, LinkedIn, etc. have started to adopt event-driven programming as an option to respond to a large number of concurrent requests and achieve scalability more operationally.[1] In this sub section, it first describes the thread-based scripting approach and its limitations. As an option to overcome limitations of thread-based scripting, it discusses the key features of the event-driven scripting approach and the benefits of the recent server-side platform based on event-driven scripting called Node.js.

[1] https://www.codefactoryacademy.com/posts

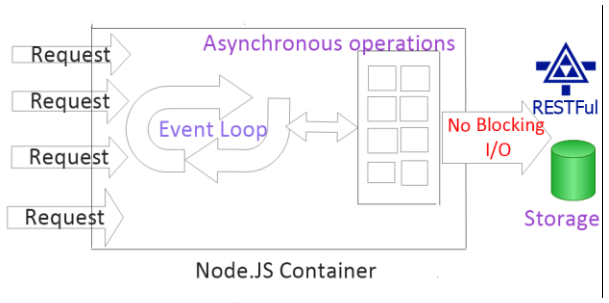


Fig. 1. A Conceptual Model for an event-driven architecture. Each incoming client request is handled by the single-thread event loop. Event handlers do trigger I/O actions that result in a new event later asynchronously.

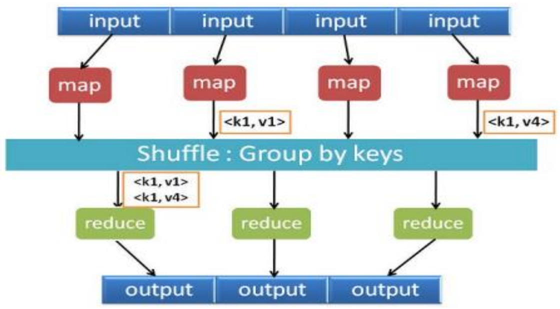


Fig. 2. Operation phases in Map/Reduce programming mode [22].

Thread based approach identities each incoming request or tasks using a separate – thread. This thread-based architecture relies on the per-connection process model where a dedicated process for handling a connection is setup [18]. The thread descriptor is shared among all processes and each process stops up for a new connection, treats the connection and then delays for the next connection [18]. As a consequence, at a given time, the server finds itself in situation having the same number of threads as the number of requests [19]. Therefore, the application server would not scale efficiently when there are many threads or network requests [19].

Event-driven scripting approach attests as an option to synchronous blocking I/O (Input/Output). As shown in Fig. 1, the event-driven approach 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. The event-driven script is referred to asynchronous programming. This means that the statements inside the scripts are not necessarily executed in the order of being written in the code. Usually, no single statement will ever block the next line of code. In fact, even if the next line statement takes a long time to complete, the rest of the program will continue to run normally. At that time, the program will wait for some resource to complete its long-running tasks and when it is done, a callback function is called. For instance, in the case of a server-side web application, this paradigm allows for handling enormous load capabilities because it does not need to wait for a long running request to finish. Instead, the server can start beginning performing the next request and return to finish the previous request when callback result occurs. The server is never blocked, so it is suitable to handle a high number of concurrent requests.

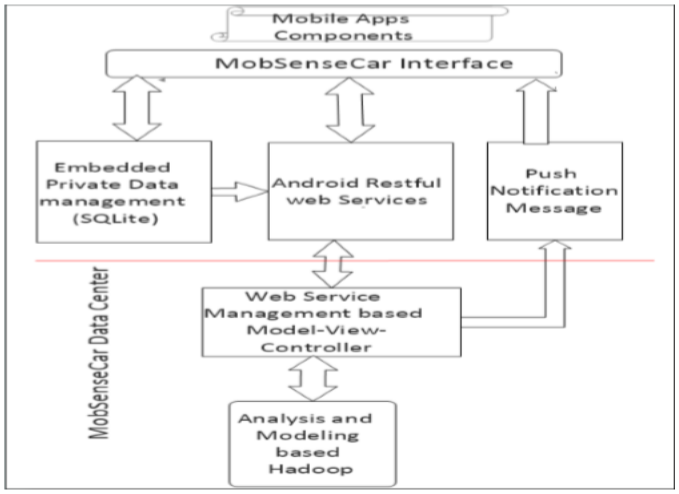

Mobile Apps Components
MobSenseCar Interface
Embedded Private Data management (SQLite)
Android Restful web Services
Push Notification Message
Web Service Management based Model-View-Controller
Analysis and Modeling based Hadoop
MobSenseCar Data Center

Fig. 3. Overview of the functional components-based Mobile Crowd Sensing and computing architecture for MobiSenseCar application. The components are divided in two type. The first type of components are designed to be implement on the Smartphone The second type of components is to manage the MobiSenseCar Data center-Wired SOA infrastructure.

One of the advantages of Node.js over thread-based framework is that it has a built-in single-thread event loop and non-blocking model [20]. The second advantage is that Node.js allows event-driven paradigm which 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 the appropriate event. The event handlers in Node.js are known as callback functions. Therefore, callback functions are eventually executed on completion of the non-blocking operation. So, when the event loop in Node.js receives the completion feedback, it executes the callbacks.

C. Apache Hadoop for Big Data

Yahoo, Google, and Facebook have extended their services to web-significance due to the amount of data collected on a daily basis. The data collected on line have overpowered the capabilities of the traditional IT architecture [21]. In order to extract the valuable data for decision making, they published open access papers and released code for core infrastructure responsible for distributed storage and processing into open source. Among these components, Apache Hadoop [22] has rapidly emerged on the top of components capable of aggregating, transforming, and analyzing server logs and a large volume of unstructured data.

Apache Hadoop is an open source distributed software platform for storing and processing data. It is written in Java, and runs on a cluster of industry-standard servers configured with direct-attached storage [21]. The structure and principal components of Apache Hadoop framework are described in details in the book “Hadoop The Definitive Guide by Tom White(foreword by Doug Cutting)” [23]. The distributed processing framework known as MapReduce (Fig. 2) is central to the scalability of Apache Hive. MapReduce helps programmers solve data-parallel problems. It splits the input data-set into multiple chunks, each of which is assigned a map task that can process the data in parallel. Map tasks functions read the input as a set of (key, value) pairs and produces a set of (key, value) pairs as result. The MapReduce framework shuffles and sorts outputs of the map tasks, forwarding the intermediate (key, value) pairs to the reduce tasks, which joins together and produces final results. MapReduce uses JobTracker and TaskTracker mechanisms to schedule tasks, monitor them, and restart any task that fails [21].

The Apache Hadoop platform also includes the HDFS (Hadoop Distributed File System). The HDFS is designed for scalability and fault tolerance. HDFS stores large files by splitting them into blocks (64MB). Beside MapReduce and HDFS, Apache Hadoop also includes many other components, some of them are very useful for analysis and modeling data: 1) Apache Flume [2] is a distributed system for collecting, aggregating, and moving large amounts of data from multiple sources into HDFS or another central data store; 2) Apache Sqoop [24], is a tool for transferring data between Hadoop and relational databases. You can use Sqoop to import data from a MySQL into HDFS, run MapReduce on the data, and export them back into a Relational Database Management System (RDBMS); 3) Apache Hive [24] and Apache Pig [24] hold programming languages that make simpler development of applications using the MapReduce framework. HiveQL is a dialect of Structured Query Language (SQL) that supports a subset of SQL as query syntax. Although slower in running, HiveQL scripts are being actively enhanced for low-latency queries on Apache HBase [24] and HDFS. In contrast, Pig Latin is a procedural programming language that provides high-level abstractions for MapReduce. Open Database Connectivity/Java Database Connectivity (ODBC/JDBC) [3] Connectors for HBase and Hive are proprietary components expected in distributions for Apache Hadoop software. They provide connectivity with SQL applications by translating traditional SQL queries into HiveQL commands that run on the data set in HDFS or HBase.

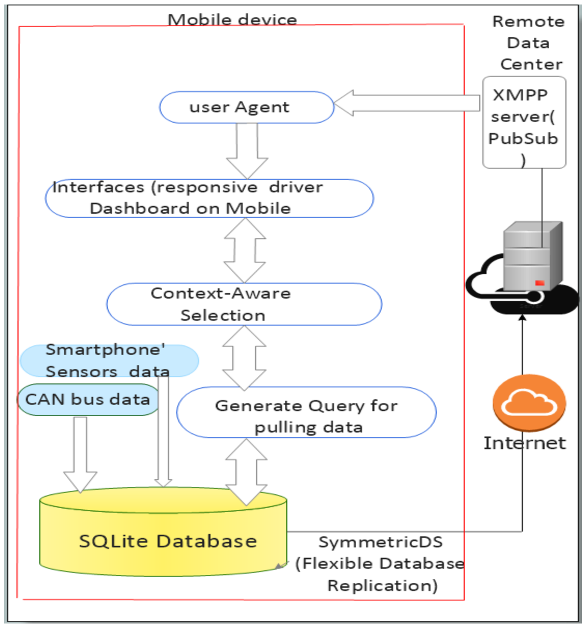


Fig. 4. Application Architecture delineating the various processing modules and interaction flow between them.

[2] Apache Flume,https://flume.apache.org
[3] Sqoop connector, https://sqoop.apache.org

IV. Design of MobiSenseCar Application for Vehicle Data Acquisition and Processing in Vehicle-Based Sensing

This section covers the key functional components of designing functional components for vehicle data acquisition and processing in vehicle-based sensing. It outlines the models for building and deploying MobiSenseCar on a smart phone within Android.

A. *Tethered Solution for the Proposed MobiSenseCar*

The design is based on the choice of database, web server architecture, the manner in which vehicle's diagnosis data are transmitted to the remote data center through a service based on Representation State Transfer (REST) called RESTful framework, and the mechanisms used for event-driven approach to handle multiple Hypertext Transfer Protocol(HTTP) requests from the MobiSenseCar application.

The tethering solution reinforce the implementation of MobiSenseCar based Mobile Crowd Sensing and Computing (MCSC) [25]. This means that the driver can access the Internet anywhere, connect to the remote data center based cloud computation and services via the internet over wireless communication technology. In this study, the tethered solution based on MCSC includes several approaches that provide the essential platforms for allowing monitoring of car's diagnostic system, thus enhancing driving convenience. The tether application on the driver's smart phone will work through normal communication cost subscription SIM to transmit vehicle's diagnosis data to the remote diagnosis data center via wireless communication technology. In order to prevent continuous transmission of vehicle's data continuously to the MobiSenseCar data center, the MobiSenseCar application would have the functionality to store the engine's status data into embedded application's SQLite database on Android [26].

The architecture of the MobiSenseCar's web service server is event-driven and can improve scalability for efficient handling of several requests simultaneously. The aim of adopting event-driven server-side architecture is to prevent blocking and long running requests that may increase cost. Certainly, the adoption of event-driven web architecture to build web-based mobile application would permit car drivers to safely open and access existing mobile apps on their smartphone while the MobiSenseCar tasks are running in background such as uploading of vehicle's diagnosis system data stored in the embedded application's SQLite database to the MobiSenseCar data center.

B. *Overview of Functional Components for MobiSenseCar*

The functional components of the architecture proposed in this study are developed around a set of high-level functional areas common to most developers of mobile application based MCSC. Fig. 3 shows the overview of the functional components-based mobile client server computing architecture for MobiSenseCar. The overview of Fig. 3 is explained as follows. First, the MobiSenseCar interface enables registration of car owner's profile and the unique identification of the OBD scan tool [4] and several services in order to enable the real

[4] ScanToolnet,https://www.scantool.net/

monitoring of the status of his (her) vehicle's engine. Second, Android RESTful web service routes the collection of vehicle's diagnosis data to the MobiSenseCar data center.

The Push Notification message handles warning notifications about OBD-II (check Engine Light) Trouble Codes. Third, the SQLite temporary stores the vehicle data while the car is driving in order to prevent the real-time transmission of this data. Many mobile applications rely on distributed key-value stores like SQLite for low latency access to data [27]. However, this SQLite database has the advantage of storage, consultation of SQLite's tables of vehicle's diagnostics data, GPS location, and notification message tables as fast as possible. Thus, drivers cannot be adversely impacted by the execution of other smart phone applications. In addition, the MobiSenseCar mobile apps has service that allows the car driver to use a non-persistent store for recording information related to its vehicle's diagnosis data such as SD memory card. This non-persistent store offers later monitoring against the corresponding fault of the vehicle's engine.

The SQLite's warning notification table maintains a mapping of unique identification-specific Id of mobile device that is used to identify the car driver for whom the warning message is concerned. The warning notification message is directly delivered using identification-specific Id of the mobile device. The GPS location table is essential when an urgent intervention is required in order to help drivers who face breakdown of their car. The collection of vehicle's diagnosis, warning message and GPS locations can be useful when evaluating the performance of vehicles sold, refining targeting, and adjusting deployment decisions on making new cars. Therefore, vehicle's diagnostics data are left entirely transient in the key-value store or kept more permanently in a movable hard disk. The decision is largely reflective to the car owner's tolerance for data loss.

On the MobiSenseCar data center-wired SOA infrastructure, Node.js web server based on the event-driven architecture interacts directly with NoSQL database (MongoDB)[5] through the framework designed for inserting and retrieving data. Hence, the effectiveness of fault detection algorithms run using MapReduce framework make it possible to analyze and process the huge volume of vehicle's diagnostics data. The storage module consists of NoSQL database such as MongoDB that avoids traditional table-based relational database structure. The particular suitability of a NoSQL database depends on the problem it is designed to manage. For big data and real-time web applications, the data structures used by NoSQL databases are also viewed as "more flexible" than the relational database tables.[6]

C. Processing Modules for MobiSenseCar

The system design focus on the implementation of a MobiSenseCar Application using OBD-II scan tool, mobile device wireless communication technology, and OEM wireless SOA infrastructure. This OEM wireless SOA infrastructure integrates web service management, and big data analytics infrastructures. Fig. 4 shows the application architecture delineating the various processing modules. Functional components of MobiSenseCar explained in the previous section constitute the basis of the proposed system design of MobiSenseCar. The user agent is responsible for controlling the whole application state and integrating push notification using XMPP server. The user agent authorizes the delivery of notification. The user agent initializes the configured smartphone's sensors. Each sensor is executed in a separate thread. Resource allocated to each sensor communicates with the corresponding sensors. The engine information and diagnostics troubles codes (CAN bus data) are temporarily stored in the MobiSenseCar application database (an android SQLite database). The context-aware selection module implemented the dynamic context-aware selection of which sensor data to be discarded. Based on the parameters from the context-aware selection module, a query is formed through it, the matched data stored on the SQLite database are replicated to the remote MobiSenseCar data center through SymmetricDS.[7] For example, when the car driver connects to the internet or activates internet connection through the smartphone's data plan, he (she) in turn transmits vehicle's engine data to the MobiSensecar data center using SymmetricDS. Therefore, in order to monitoring the car's diagnostics system continuously and seamlessly, a web server-side solution based on Node.js enables the availability of vehicle's engine data for sharing purposes. The Node.js is responsible for handling data transmitted to a NoSQL database (MongoDB database).

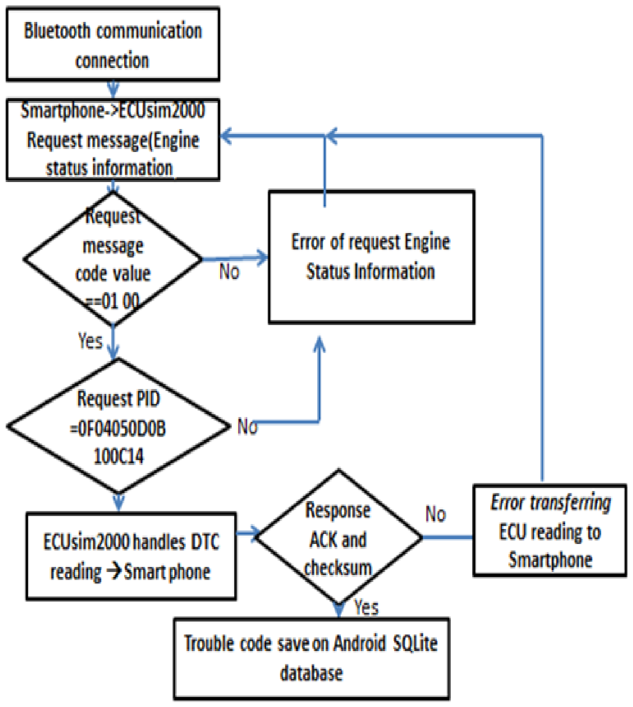

Bluetooth communication connection
Smartphone->ECUsim2000 Request message(Engine status information
Request message code value ==01 00
No
Yes
Error of request Engine Status Information
Request PID =0F04050D0B 100C14
No
ECUsim2000 handles DTC reading → Smart phone
Response ACK and checksum
No
Yes
Error transferring ECU reading to Smartphone
Trouble code save on Android SQLite database

Fig. 5. Flowchart of engine status information collection algorithm.

[5] https://github.com/mongodb/mongo-hadoop/releases
[6] http://www.allthingsdistributed.com/2012/01/amozon-dynamodb.html
[7] https://www.itcentralstation.com/comparisons/ibm-infosphere-database_vs_jumpmind-symmetricds

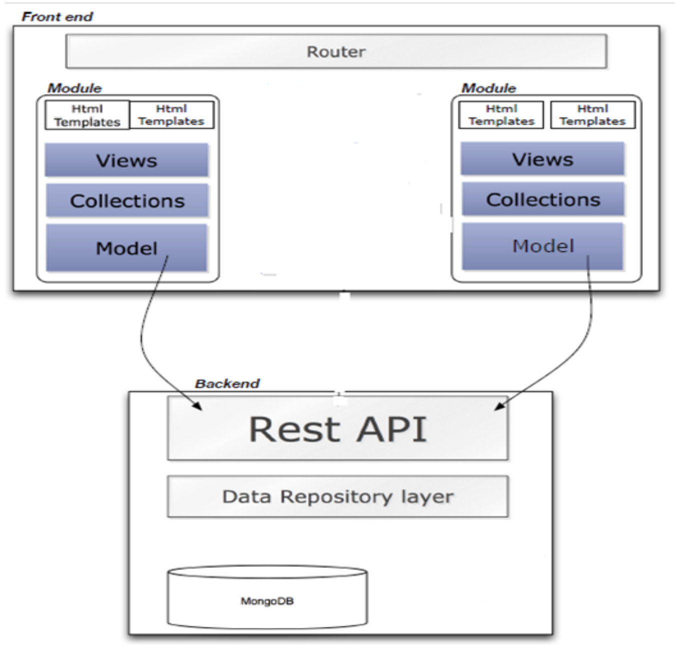


Fig. 6. Main Software Components of handling vehicle-based sensing data on the MobiSenseCar data center.

D. *Bluetooth OBD-II Protocol Structure for Engine Status Information Collection*

The system design of the MobiSenseCar service allows the driver confirming the status of engine in real-time via his (her) mobile device or web browser using a portal web site designed to retrieve the current vehicle's status information. Drivers may always check the status of the engine with a smart phone. For business purpose with the car manufacturer and vehicle commercial companies, the driver can transfer the current vehicle's status information to the MobiSenseCar data center. The engine status information collection algorithm is shown in Fig. 5.

In this study, it used an On-Board Diagnostic simulator called OBD-II simulator (ECUsim2000). [8] With this ECUsim2000 simulator, the 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 speed, revolutions per minute (RPM), intake temperature, coolant temperature. As shown in Fig. 5, the proposed system architecture includes Bluetooth communication between the ECU simulator (ECUsim 2000) and the Bluetooth interface that supports all OBD-II protocols and receiver (mobile) devices. The ECUsim2000 is designed to ensure integrity of supporting all OBD-II protocol in order to read OBD-II parameters clearly. Hence, the flow of data acquisition is followed by receiving byte for OBD-PID request when the message address matches. Subsequently, the value of request OBD-II parameter is transmitted via Bluetooth communication. At the receiver side, a smart phone plots the received engine's diagnostics data. The different value of OBD-II parameters are calculated in a human readable form and displayed on the MobiSenseCar mobile application console designed for displaying current OBD-II parameters read out. The OBD-II message format consists of 1-byte priority, target address, source address header, 7 byte data, and checksum. It is basically used as a protocol for SAE-J1850 and ISO [28]. The CAN OBD message format consists of ID bits (11or 29), DLC, 7 data bytes, and checksum[9] (CRC-15 processing method).

E. *Deployment of Web-Side Based Node.js and MongoDB Database for Handling Vehicle-Based Sensing Data*

The Node.js presents several advantages in terms of processing multiple connections or tasks concurrently. In our study, Node.js has influenced our choice of the web server architecture. Therefore, the MobiSenseCar application consists of collecting the vehicle sensing and transferring them to the remote MobiSenseCar center for further processing. The car's diagnosis system data need to be stored or hosted on a computer which is connected to the internet known to us as a Web Server. It serves to handle incoming requests from the web browsers (clients), and then responds by sending the required data through a web server program. As shown in Fig. 6, the router is the component that organizes routing between theMobiSenseCar's main pages. The router is configured to listen to every event so that when such an event is triggered, the router is notified which in effect tells to issue the URL request to the server. The models implement both business logic and data attributes.

Considering that the application has several collections of models, for example a list of current CAN bus data on the vehicle sensing data page, multiple rows of CAN bus data in the vehicle sensing data page, etc. It makes sense to encapsulate these models in separate modules (collections). This way, a collection is a holder for multiple coherent models. The motivation for this is that the collections can also work as active records, in that they can be responsible for fetching and maintaining a particular set of vehicle sensing data or CAN bus data from the database, regarding the collection of models they control. The view handles all the user-interactions that happen inside the HTML, it represents. The view is responsible for rendering its HTML templates.

Let now see how Node.js handles blocking I/O requests since MobiSenseCar application involves an important number of concurrent requests. These requests consist of transferring car sensing data such as vehicle's diagnostic data to the MobiSenseCar center. The Node.js event model uses event handler. This event handler comes on with great results until you run into the development of functions that involve blocking I/O. The blocking I/O is defined as a request that stops the execution of the current thread and delays for a response before continuing. With Node.js, the event model work is scheduled in advance as a function with a callback to the event queue.

MongoDB is a technology that is revolutionizing database usage. Together, the two tools (Node.js and MongoDB) are a powerful combination to the fact that they both employ JavaScript and JSON. We will need to install Mongoose, which is the library that Node.js uses to communicate with

[8] https://www.scantool.net/ecusim-2000.html

[9] http://smartdata.usbid.com/datasheets/usbid/2000/2000-q4/j1850 wp.pdf.

MongoDB. The web service functions on Node.js would collect data from the vehicle when the OBD connector is paired with the MobiSenseCar application based on Android platform. Thus, through 3 or 4G LTE and HTTP protocols, data are transferred over the internet in JSON exchange format for easily processing on the web server with Node.

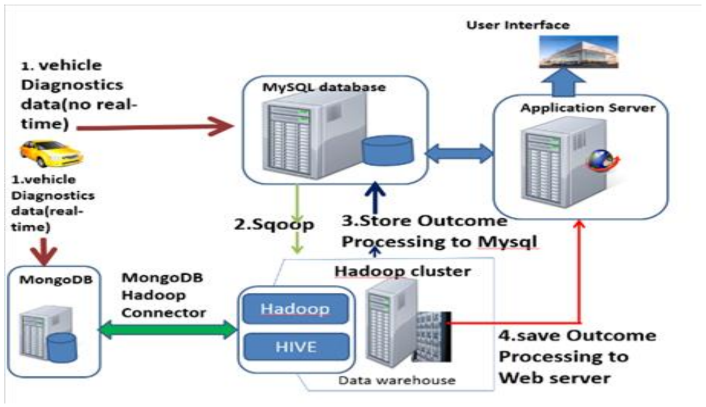

User Interface
1. vehicle Diagnostics data(no real-time)
MySQL database
Application Server
1.vehicle Diagnostics data(real-time)
2.Sqoop
3.Store Outcome Processing to Mysql
Hadoop cluster
MongoDB
MongoDB Hadoop Connector
Hadoop
HIVE
4.save Outcome Processing to Web server
Data warehouse

Fig. 7. Architecture and components of analysis model Based Big data analytics.

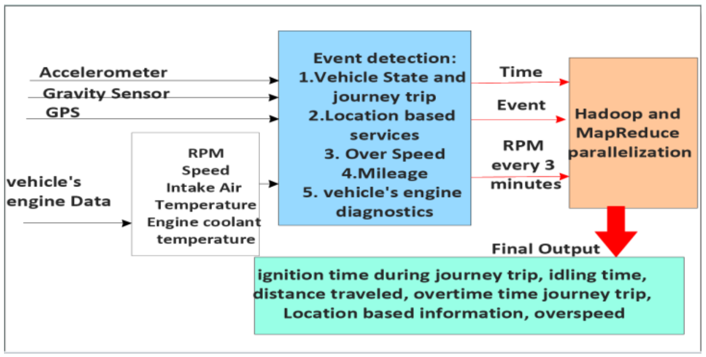

Accelerometer
Gravity Sensor
GPS
vehicle's engine Data
RPM
Speed
Intake Air Temperature
Engine coolant temperature
Event detection:
1.Vehicle State and journey trip
2.Location based services
3. Over Speed
4.Mileage
5. vehicle's engine diagnostics
Time
Event
RPM every 3 minutes
Hadoop and MapReduce parallelization
Final Output
ignition time during journey trip, idling time, distance traveled, overtime time journey trip, Location based information, overspeed

Fig. 8. Design of processing model for Monitoring and Analysing Vehicle' data based sensing using Hadoop and MapReduce parallelization.

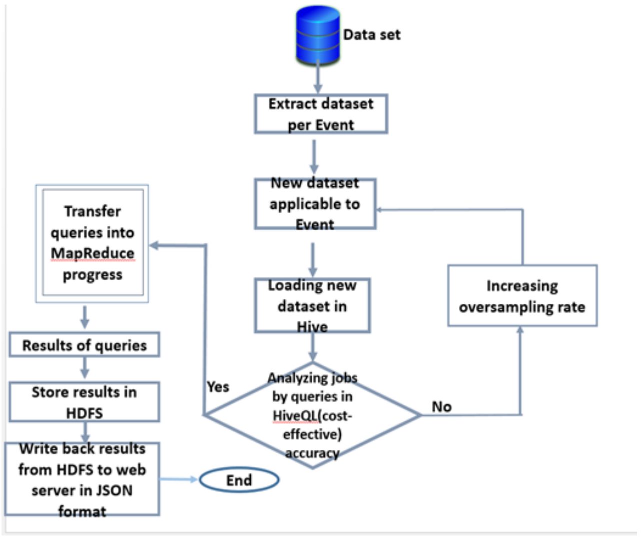

Data set
Extract dataset per Event
New dataset applicable to Event
Loading new dataset in Hive
Analyzing jobs by queries in HiveQL(cost-effective) accuracy
Yes
No
Increasing oversampling rate
Transfer queries into MapReduce progress
Results of queries
Store results in HDFS
Write back results from HDFS to web server in JSON format
End

Fig. 9. Flow chart of the procedure followed during the implementation of the processing model of vehicle-based sensing data.

F. Formal Representation of the Proposed Analysis Model Big Data and its Architecture

The proposed analysis model using Hadoop framework is based on a data-driven approach. This approach consists of collecting data sets uploaded from vehicles. Those data are then monitored based on different aspects of activity of the vehicles that we quote as “Events”. The first event relates the vehicle’s movement and journey trip. The second event is the collection of vehicle’s diagnostic data while the driver is driving.

The processing of the data acquired from the MobiSenseCar is divided into four phases as shown in Fig. 7:

1) Import data from MySQL to Hadoop clusters.

2) Loading data from HDFS to Apache Hive.

3) Analysis through Hadoop MapReduce framework.

4) Upload outcome files in JSON format from HDFS to the web server.

Firstly, the import data from MySQL to HDFS consists of importing data from MySQL into Sqoop. Sqoop is an open-source tool that allows users to extract data from a relational database into Hadoop for further processing [24]. The MongoDB Connector for Hadoop provides the ability to use MongoDB as input and/or an output destination [29]. Secondly, the loading data from HDFS to Hive is performed by Apache Sqoop. It keeps parallelizing import across multiple mappers. The import data into Apache Hive relies on the sake of efficiency that has a post processing step where Hive table is created and loaded. When the data is loaded into HIVE from HDFS directory, Hive moves the Sqoop replication table which is viewed as a directory into its warehouse rather than just copying data. Thirdly, Hive queries feature join patterns algorithms are implemented in the MapReduce jobs to execute SQL applications and queries [29] and finally, the Apache Sqoop writes back the output results to MySQL Database or MongoDB.

The description model of the proposed analytics framework associates for both events includes the vehicle’s movement, journey trip, location based service, over speed, mileage and diagnostics of vehicle’s engine events which is an appropriate subset of information. The subset of information are journey data, Global Positioning System (GPS) data, driver behavior data based on the smartphone’s in-built accelerometer, engine data and car diagnostics data. As shown in Fig. 8 for instance, the analysis process takes the vehicle’s movement and journey trip event, and then associates the RPM value of the vehicle to detect if the engine is running, current data, and accelerometer data to detect vehicle’s movement. For this event, the analysis process extracts the value of the RPM every three (3) minutes to detect the state of the vehicle, which is either in idle state or not. These data are then uploaded to HDFS. The data set on the HDFS serve as the basis for collecting useful information to submit to MapReduce functions for processing.

The Hadoop framework splits the input data-set into multiple chunks, each of which is assigned a map task that can process the data in parallel. Each map task reads the input as a set of key-value pairs and produces a transformed set of key-value pairs as the output. The framework shuffles and sorts outputs of the map tasks, and send the intermediate key-value pairs to the reduce tasks, which group them into final results. For the vehicle's movement and journey trip the results are for example vehicle movement time, idling time, traveled time, and journey trip time. These results are then stored on the hosting database with an additional field to indicate on which vehicle's telematics applications the output are intended to be applied .

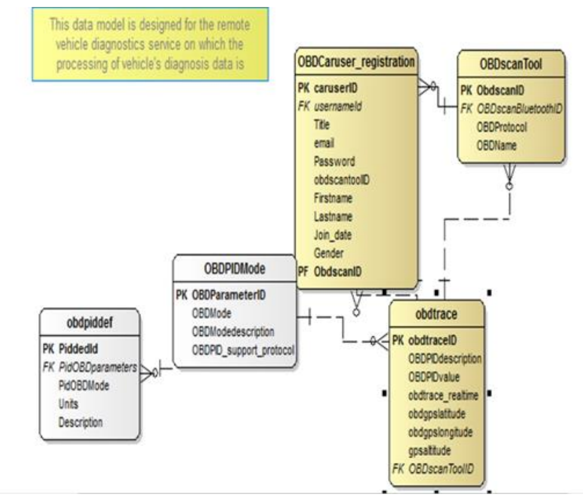
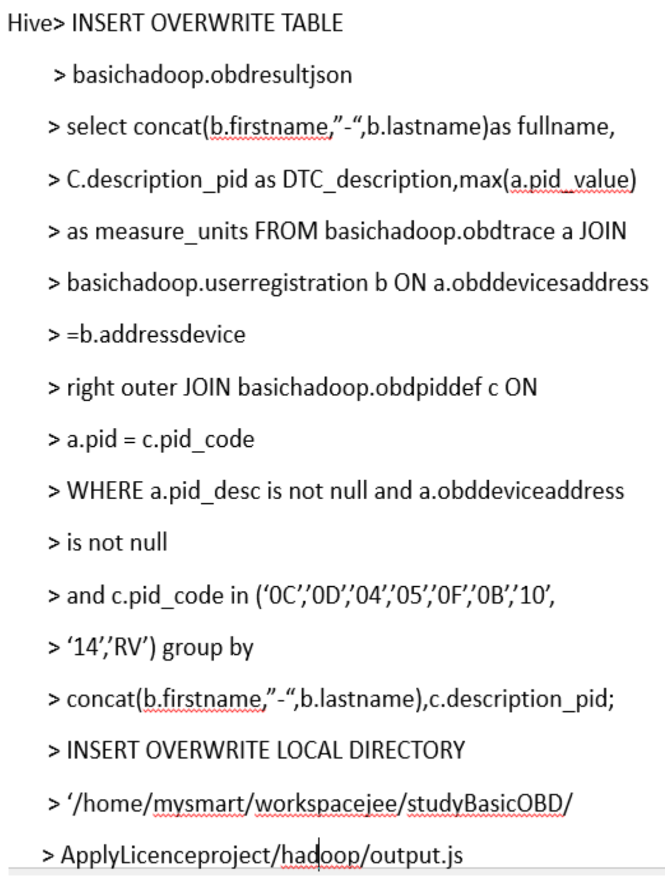


Fig. 10. Data model that includes OBD-PID II description, car user information, and the current vehicle data collected from vehicle.



```
Hive> INSERT OVERWRITE TABLE
   > basichadoop.obdresultjson
  > select concat(b.firstname,”-“,b.lastname)as fullname,
  > C.description_pid as DTC_description,max(a.pid_value)
  > as measure_units FROM basichadoop.obdtrace a JOIN
  > basichadoop.userregistration b ON a.obddevicesaddress
  > =b.addressdevice
  > right outer JOIN basichadoop.obdpiddef c ON
  > a.pid = c.pid_code
  > WHERE a.pid_desc is not null and a.obddeviceaddress
  > is not null
  > and c.pid_code in (‘0C’,’0D’,’04’,’05’,’0F’,’0B’,’10’,
  > ‘14’,’RV’) group by
  > concat(b.firstname,”-“,b.lastname),c.description_pid;
  > INSERT OVERWRITE LOCAL DIRECTORY
  > ‘/home/mysmart/workspacejee/studyBasicOBD/
 > ApplyLicenceproject/hadoop/output.js
```

Fig. 11. HIveQL script for analyzing the current value of OBD-II parameters.

Fig. 9 presents a flow chart of the procedure followed during the implementation of the processing model for vehicle-based sensing data. The in-vehicle's diagnostic data is stored in HDFS; analyzing jobs are executed by queries in HiveQL language, and the queries will be transferred into Mapreduce progress. The results of queries are stored in HDFS as well. Thus, the result in Hive shall be transported from Hadoop to the web server. The aggregation analysis is based on the join of two tables, the "obdtrace" and the "obdpiddef". The data model of the MobiSenseCar application is shown on Fig. 10. The aggregation work is realized by integrating these two tables into Hive script is shown in Fig. 11.

V. Implementation and its Results

In this section, it presents a performance measurement of the proposed design for implementing the MobiSenseCar mobile application. First, it presents the performance of Client-Server Architecture that features event-driven approach against thread-based approach at the sever-side layer. The goal of this measurement is to investigate which of the Client-Server Architecture configurations is able to support concurrent clients' requests. Secondly, it evaluates the efficiency of Hadoop MapReduce computing using join reduce side algorithm against HiveQL and Statistical Analysis System (SAS) framework [30].

To ensure integrity and reliability of the proposed mobile crowdsensing architecture for MobiSenseCar, different experiments are carried out to collect results. A physical real-time monitoring experiment is performed to ensure data transmission from the ECUsim 2000 equipped with Bluetooth to the mobile device. A MobiSenseCar application is built to assist the car owner in his daily monitoring of engine.

The engine's data synchronization and MobiSenseCar computing system are tested during the experiment to collect all the OBD-II data and seamlessly track the engine status. An event-driven web server Node.js platform is used to run JavaScript outside the browser. A MobiSenseCar Web page is developed for easy monitoring purposes. Finally, the simulation results for evaluating the performance of client-server architecture to handle concurrent requests as well as the performance of the MapReduce against traditional data processing framework are discussed in the second part of this section. Hadoop platform is used to analysis this engine's data and handling the result.

In order to evaluate the collection of car engine data of MobiSenseCar application based on Android's mobile device, the experimentation environment consists of three main components. First, Android mobile device version 4.4.2 is used in the implementation. Second, web server with Node.js, a core i3-3220 CPU within 3.3GH, 16 GB of RAM running Windows seven 64 bit and finally Hadoop multi-node cluster in a distributed environment using three systems (one master and two slaves , each of them is a core i5-6600Processor within 3.90GHz, 16 GB of RAM). Such an environment facilitates to run the collected data from vehicle-based sensing on a real cluster of servers.

A. Performance Measurement of Event-Driven Approach With Node.Js Against Thread-Based Approach with Apache

To measure the performance of event driven approach, the program Tsung was used.[10] The component under test was the main back end server. Tsung works by simulating multiple users making multiple requests to the server. Every user is run in a separate thread. To simulate normal conditions, Tsung allows user think-time and the arrival rate to be specified using a probability distribution. Several tests of each scenario were carried out under different cluster of client-server configurations and varying amount of loads generated by Tsung.

The test plan uses TSUNG 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 are required to show that a Client-Server Architecture consists of backend and database layers can run with acceptable responsiveness when a large number of concurrent users access the backend server-side and database simultaneously.

This study has considered three kinds of Client-Server Architectures that provides the 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 for use case. Thus, MongoDB fits perfectly for Node.js applications. Therefore Node.js and MongoDB allows writing JavaScript for the backend and database layer [31]. Furthermore, MongoDB is known for its schemaless nature that gives a better way to match the constantly evolving data structures in the MobiSenseCar application. 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 written using JavaServer Pages (JSP) technology [32]. 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, it uses the Java API for MongoDB/BSON in Apache Tomcat [27]. For each of the Client-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 based on the mobile client server computing that has a module of uploading the vehicle diagnostic data stored in a temporary data store like SQLite to the remote application server.

The test environment for client-server Architecture 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:

1) A user http request arrives over a SSL to the application server.

2) The application server forward calls to the server in the web tier.

3) The web tier runs in a computer 3.30 Ghz Intel core i3, 8GHz on Windows server.

4) The web tier runs Node.js server, Express for Node.Js and code of MongoDB object modeling for Node.js.

5) 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.

After defining the test environment for the three Client-Server Architectures for the MobiSenseCar server, it assumes that the goal of the tests is to establish the capacity of a server for handling vehicle diagnostics data from the mobile device by a large number of concurrent users starting simultaneously. Each Client-Server Architecture supports the excepted peak load of concurrent users.

Tsung runs these tests locally from a different computer running 2.2GHz Intel core i3, 4GB DDR on OS windows seven. Scripts are created to determine the supported number of concurrent requests of uploading of vehicle diagnostics data and to simulate the concurrent users sessions.

The first tests against Node.Js server side and the Apache Tomcat server were conducted simultaneously from 200 users up to 2000 users, 50 times. For each test, the "ramp-up" period is zero (0) which means that all the users start sending the http request simultaneously. We do however need to understand the capacity of the client-server architecture such that we can determine at what point uploading of vehicle diagnostics is completed successfully by a large number of concurrent users. The peak load testing scenarios state are shown in Table 1.The experiments requirements are defined in Table 2.

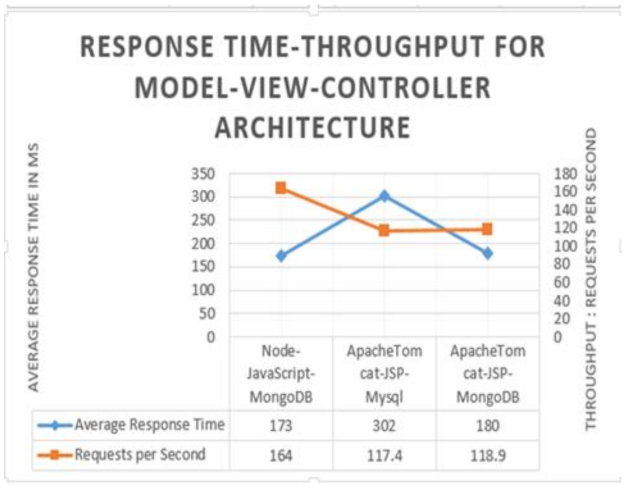
TABLE I. SCENARIOS CASES FOR THE EXPERIMENTATION

#Scenarios	**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
Scenario 4	800	200	50

TABLE II. REQUIREMENTS FOR SCENARIOS CASES FOR THE EXPERIMENTATION

Requirements for scenario 1, 2 and 3	**Value**	**Additional information**
Volume of load : upload from 200 users up 2000 users at the same time	Peak load : 1 scenario iteration every 0 second , 50 times	Equates to 100000 samples
How long should load be run	Less than 5 minutes	
Acceptance criteria	95% stored in the corresponding database successfully	
Metrics to be reported on	Response time, throughput, error rate	DB activity

[10] http://tsung.erlang-projects.org



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

Fig. 12. The performance of three model-view-controller based on the throughput and response time metrics.

Fig. 12 shows the results of the three Client-Server Architecture configurations. To this performance, the results help for analyzing 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 includes Apache at server-side and MongoDB as database outperforms less better in comparison to Node.js-JavaScript-MongoDB but better than Apache Tomcat-MySQL.

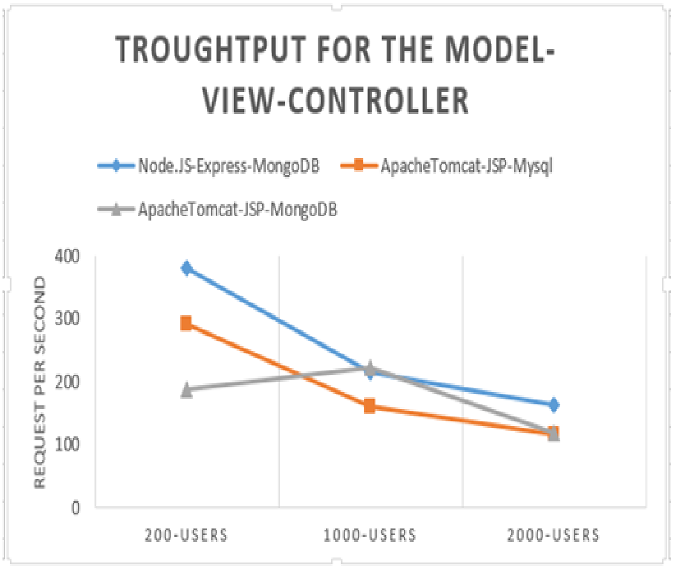


Fig. 13. Throughput of the three model-view controller for n concurrent users.

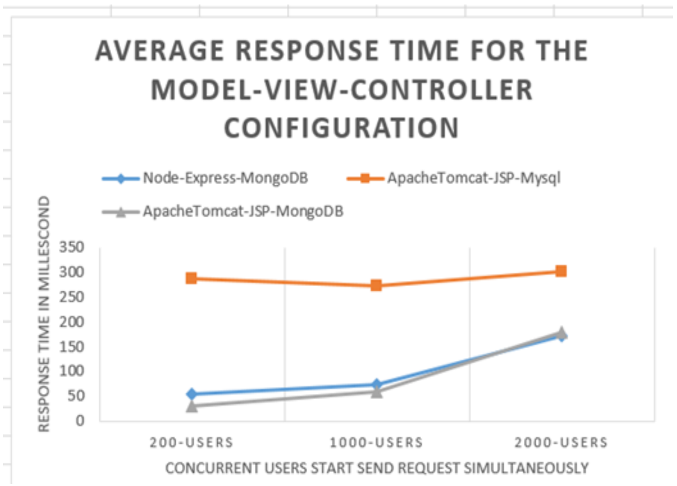


Fig. 14. The measurement of the response time on average for the three-view-controller architecture as the number of concurrent users increase.

Fig. 13 shows that the throughput deteriorates for the three models as the number of concurrent users increases. The results show that for Node.js associated to MongoDB, the throughput was 380 requests at 200 concurrent users, and 164 requests per second at 2000 concurrent users. The throughput of Node.js based event-driven approach has performed better than Apache-Tomcat based thread approach. The results were better using Apache-Tomcat-MongoDB than Apache Tomcat-MySQL and less good than Node.js-Mongo. With Apache-Tomcat-MongoDB configuration, the throughput increases up to 1000 concurrent users, then deteriorates as the number of concurrent users increases. There may be some tremendous opportunities for optimization that could enforce Node.js-MongoDB performance beyond Apache-Tomcat-MongoDB easily.

As shown in Fig. 14, the response time degrades as the number of concurrent requests increases. For example, Node.js-MongoDB was within a response time of 54ms on an average at 200 concurrent users, and 173ms on an 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 thousand-fold increase in concurrent users' results in a hundredfold increase in response time. This 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, 173ms against 302ms for 2000 users that corresponds to one hundred thousand (100000) concurrent requests.

B. *Performance Measurement of Big data Analytics Framework for Processing Vehicle's Diagnostics Data*

The performance evaluation of Big data analytics uses representative benchmarks that perform on the datasets from the MySQL tables and MongoDB. The set up experiment environment constitutes of a Hadoop multi-node cluster on a distributed environment using three systems (one master and two slaves, each of them is a core i5-6600Processor within 3.90GHz, 16 GB of RAM). Such an environment facilitates to run the remote vehicle diagnosis event processing on a real cluster of servers. Diagnostic Trouble code (DTC) is collected into a relational database, and unstructured data are stored in NoSQL database (MongoDB), and then dumped directly into Hadoop cluster. When on-board diagnostics data are uploaded onto the database, Apache Sqoop performs a replication import of data required to run Map Reduce jobs.

When on-board diagnostics data are uploaded to the database, Apache Sqoop performs a replication import of data required to run Map Reduce functions. HIVE has an important role especially for data stored in a relational database. Sqoop generates a Hive table based on the table originally relational data source and also stores data on HDFS.

One of the most key Hadoop jobs in this study is to take the incoming on-board diagnostics and summarizes according to the useful information (see Fig. 8). It stores the processing outcome to the MySQL database or copies it into a format that can be used for further analysis or purposes on the web services. This is achieved by using HIVEQL and Map Reduce functions.

In order to compare the efficiency of Hadoop MapReduce computing using join algorithm particularly reduce side join [33], HiveQL, a higher-lever framework in which join are integrated in their implementation and the traditional statistical analysis system (SAS) is used. This SAS ® 9.4 SPD Engine is used for storing data in the HDFS. It also has procedures that can replicate MapReduce's approach for data processing. Since data are in place, map reduce functions converted from HIVEQL can start analyzing them and turn processing outcome into valuable information.

The evaluation computes as well the same statistical values (mean) on Diagnostic Trouble codes stored in MySQL. Fig. 15 shows the processing time of three methods.

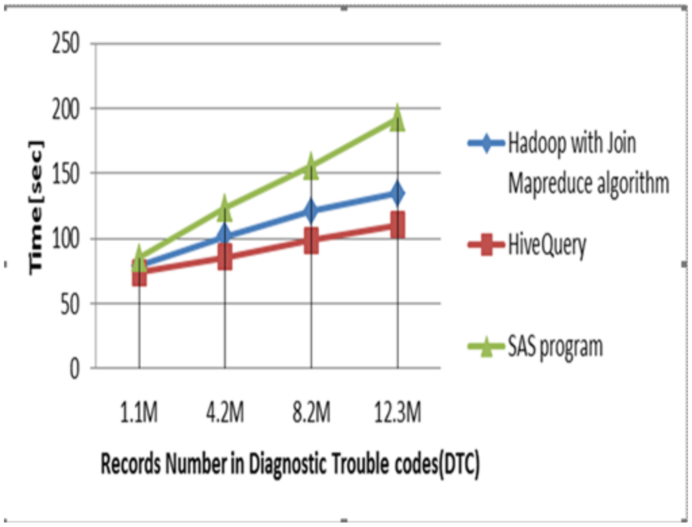


Fig. 15. Processing time of MapReduce framework features multi-way join based reduce-side cascade join.

Fig. 15 comes out with the following observations:

- Comparing to traditional statistical analysis system, Hadoop distributed parallel computing enhances processing speed when the size of dataset to be processed increases and is not significantly different over a lower volume of dataset.
- The Comparison of join algorithms using MapReduce framework to HiveQL, showed that join patters based MapReduce increases computing speed over HiveQL but it takes time to implement. HiveQL is arguably one of the tools for developers and analysts with strong SQL skills but SQL is not suitable for every big data problem.

C. *Performance Measurement of Big data Analytics Framework for Processing Vehicle's Diagnostics Data*

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 study there were different concurrency models implemented using single-threaded event loop Node.js and multi-thread approach. To test whether Node.js is a higher concurrency level-where it is supposed to surpass multi-threading, other problems like increasing the number of requests. The reason is that Tsung is a 100% pure Erlang to evaluate the functional behavior and measure performance of the three Client-Server Architecture configuration. The experimentations were not able to run these tests beyond 4000 concurrent users, over 200,000 requests.

These study findings have shown what mobile application can do with RESTful responses and requests over a web server. Node.js is considered to perform greatly. In the future it should be taken in consideration for remotely monitoring applications based on mobile device where devices spread in different locations collect a large volume of information. While Node.js outperformed the competition in the scenario of transferring vehicle's diagnosis data, further work can point Node's strengths and weaknesses. Node accomplishes its goals of supporting highly-scalable and reliable web servers. It runs very quick on JavaScript engine. Therefore, Node.js is not designed to stand simply as a replacement of Apache.

Besides using Node.js for handling asynchronous I/O requests, Hadoop stands as a new form for processing a huge amount of data from all car owners subscribed to the MobiSenseCar application. Therefore, compare to traditional statistical analysis system, Hadoop distributed parallel computing enhances processing speed when the size of dataset to be processed increases and is not significantly different over a lower volume of dataset. Thus, comparing data join MapReduce algorithm to HiveQL, relational data join patterns in MapReduce increases computing speed over HiveQL but it takes time to implement. HiveQL is arguably one of the tools for developers and analysts with strong SQL skills but SQL is not suitable for every big data problem [24].

VI. Conclusions and Future Work

This paper argues the concerns about uprightness of data from vehicle-based sensed data are a major step for vehicle owners, authorities and businesses looking to take up mobile crowdsensing computing that enables value–added and others services. This paper presented a mobile crowd-based architecture which enables car's diagnostic system that features data remote monitoring event processing of vehicle's engine data.

The system design in this paper consists of MobiSenseCar application, an event-driven web server known as Node.js, and Hadoop platform. The MobiSenseCar allows the collection of vehicle's engine data and storage into an embedded application database known as SQLite. At the MobiSenseCar data center infrastructure, the web server Node.js enables MobiSenseCar mobile application requests to be processed asynchronously. Therefore, the request of transmitting vehicle's engine data is not interrupted when a large number of car driver start using the same application simultaneously. The use of Node.js helps to save communication cost and enables car drivers to use other mobile apps on their smartphone when transmitting vehicle's engine data. The Node.js has API (Application Programming Interface) which interact for instance with a NoSQL database (MongoDB).

Taking advantage of reducing communication cost, driver would still use their smartphone for other purpose while the Node.js handles and stores the generated car's status information to a MongoDB without waiting for the MobiSenseCar application to finish the uploading process submitted to Node.js. Thus, processing data from vehicles using Hadoop and making final results available, allows accessing of useful information via web services to the third party such as car manufacturer, transportation and road operators, car dealers, police, emergency services has been conducted on a single node cluster.

The outcome obtained from various Map Reduce functions managed after executing HIVEQL query indicate favorable results in term of time taken. It is unnecessary to receive and deal with all the data including even needless one, so the car owner may handle information satisfying car manufacturers or car repair shop needs only. Therefore, with this system, it was made possible that information of car's diagnostic system condition may be identified in real time

The proposed solution leverages the existing mobile crowdsensing architecture for data collection and processing on vehicle-based sensing. There are still several challenges that must be addressed for this kind of deployment model can be adopted. As future work will focus on the deployment and empirical validation for MobiSenseCar architecture with specific focus on the collection of vehicle-based sensed data stored in real time on the cloud computing based infrastructure as a Service.

Acknowledgment

This research was supported by the Brain Busan 21 Project in 2017 and The Human Resource Training program for Regional Innovation and Creativity through the Ministry of Educational and National Research Foundation of Korea (NRF-2015H1C1A1035898).

References

[1] K. G. Raghu., Y. Fan, and, L. Hui, "Mobile Crowdsensing: Current State and Future Challenges", IEEE Communications Magazine , pp.32-39, 2011

[2] T .Carlos, C. Celimuge, W. N.Enrico, and J. Francisco, "Crowdsensing and Vehicle-Based Sensing", Mobile Information Systems,2016, pp. 1-2.

[3] W.Jinn, C. Jinsong, and M.Tinghuai, "Real time services cloud computing enabled vehicle networks". International Conference on Wireless Communications and Signal Processing (WCSP), 2011,pp: 1-5.

[4] I.E.Apetri, , R.Ali, "Remote Connection of Diagnostic Tool", Master of Science Thesis in Communication Engineering", 2011, pp:1-104

[5] M.Emanuele , A.Chaewon, and R.Carlo, "The Car as an Ambient Sensing Platform", Proceedings of the IEEE, Vol.105,No.01,2017,pp: 3-7

[6] 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.

[7] S.Stephan, G.Eszter, R. Wolfgang, "Performance investigation of selected SQL and NoSQL databases", AGILE 2015-Lisbon, 2015,pp:1-5,

[8] V.Madhavi, , "Survey of Parallel Data processing in Context with MapReduce", AIAA 2011, Computer Science & Information Technology 03(CS & IT), 2011,pp:69-80

[9] M.Prashanth,, N.P.Venkata and R.Ramachandran, "Nericell: using mobile smartphone for rich monitoring of road and traffic conditions",Proceedings of the 6th ACM Conference on Embedded network sensor system,2088,pp:357-371

[10] J.Eriksson, L.Girod, and B.Hull, "The Pothole Patrol: Using a Mobile Sensor network for Road surface Monitoring", Proceedings of the 6th ACM Conference on Embedded network sensor system, 2008 , pp:29-39.

[11] A.J.Derick and M.T.Mohan, "Driving Style Recognition Using a Smartphone as a Sensor platform",14th International IEEE Conference on Intelligent Transportation Systems, 2011,pp:1609-1615

[12] H.Eren, S.Makinist,,F.Akin., and A.Yilmaz, "Estimating Driving Behavior by a smartphone", 2012 Intelligent Vehicles Symposium,2012,pp:234-239

[13] J.Eriksson, L.Girod, and B.Hull, "The Pothole Patrol: Using a Mobile Sensor network for Road surface Monitoring", Proceedings of the 6th ACM Conference on Embedded network sensor system,2008,pp:29-39.

[14] W.Jules, , T.Chris., T.Hamilton, D. Brian, and C.S.Douglas, "Mobile Networks and Applications, Vol.16,No.3,2011,pp:285-303

[15] C.German, D.Thierry, F.Raphael, and E.Thomas, "Driver Behavior Profiling Using Smartphones : A Low-Cost Platform for Driving Monitoring", IEEE Intelligent Transportation Systems Magazine,2015,pp:91-102

[16] An Oracle White paper, "Oracle for the Connected Vehicle:Turning Data into Business",2013,pp:1-21.

[17] GsmamAutomotive, "Connecting Cars:Bring your Own Device-Tethering Challenges". Report on Intelligent Trasporatation system Report, 2013,pp: 1-20.

[18] B.Erb , "Concurrent Programming for scalable Web Architecture",Diploma Thesis, Institute of Distributed Systems,2012

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

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

[21] C.Bin., M.Hong, C.Beng, "Big data: the driver for innovation in databases", National Science Review,Vol.1,No1,2014,pp:27-30

[22] D.B.Arantxa., O.Aisling. "A big data methodology for categorising technical support requests using Hadoop and Mahoot". Journal of Big Data, Vol.1,No1,2014,pp:1-8

[23] T. White, "Hadoop: The definitive Guide, Third Edition", pp:17-44

[24] S.P.Poonam, N.P.Rajesh., "Survey Paper on Big Data Processing and Hadoop components".International Journal of Science and Research(IJSR),Vol.3,No10,2014,pp:585-590

[25] G.Bin, W.Zhu, Y.Zhiwen, G.Yu, Y.Neil,,H.Runhe, and Z.Xingshe, "Mobile Crowd Sensing and Computing : The Review of an Emerging Human-Powered Sensing Paradigm", ACM Computing Surveys, 2015, pp: 1-33

[26] B. Oresti., V.Claudia, D.Miguel, G.Peter, P.Hector, R.Ignacio, "PhysioDroid: Combining Wearable Health Sensors and Mobile devices for a Ubiquitous, Continuous, and Personal Monitoring", The Scientific World Journal Soc, Article Id 490824, 2014, pp-1-14.

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

[28] K.Minyoung, andJ. Jang-Wook," Design of Korea smart car driving information checking system". International Journal of Advanced Smart Convergence.1(1),2012,pp:38-42.

[29] B.Mani, J. Balaraj, M.D.Oinam, "Comparison of Join Algorithms in MapReduce Framework",International Journal of Innovative Research in Computer and Communication Engineering,Vol.2,Special Issue 5, 2014

[30] M.David, SASReduce-An implementation of MapReduce in BASE/SAS. Whitehound Limited, UK. Online athttp://support.sas.com/resources/papers/proceedings14/1507-2014.pdf, (2014), 1-16.

[31] K.Brian,, "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

[32] L.N.Glenn, "Tomcat Performance Tuning and Troubleshooting",ApacheConference, pp:1-10,2003

[33] N.A.Foto, and D.U.Jeffrey, "Optimizing Joins in a Map-Reduce Environment",IEEE Trans.Knowl.Data Eng.23(9):2011,pp1282-1298