Hindawi Publishing Corporation
Journal of Sensors
Volume 2015, Article ID 834217, 11 pages
http://dx.doi.org/10.1155/2015/834217

Research Article

Architecture and Implementation of a Scalable Sensor Data Storage and Analysis System Using Cloud Computing and Big Data Technologies

Galip Aydin, Ibrahim Riza Hallac, and Betul Karakus

Computer Engineering Department, Firat University, 23100 Elazig, Turkey

Correspondence should be addressed to Galip Aydin; gaydin@firat.edu.tr

Received 6 February 2015; Accepted 20 February 2015

Academic Editor: Sergiu Dan Stan

Sensors are becoming ubiquitous. From almost any type of industrial applications to intelligent vehicles, smart city applications, and healthcare applications, we see a steady growth of the usage of various types of sensors. The rate of increase in the amount of data produced by these sensors is much more dramatic since sensors usually continuously produce data. It becomes crucial for these data to be stored for future reference and to be analyzed for finding valuable information, such as fault diagnosis information. In this paper we describe a scalable and distributed architecture for sensor data collection, storage, and analysis. The system uses several open source technologies and runs on a cluster of virtual servers. We use GPS sensors as data source and run machine-learning algorithms for data analysis.

1. Introduction

Sensors are generally used for measuring and reporting some properties of the environment in which they are installed, such as the temperature, pressure, humidity, radiation, or gas levels. Traditionally these measurements are collected and stored in some sort of a data store and then are processed to find any extraordinary situations. However in such cases like smart city applications where large numbers of sensors are installed, the amount of data to be archived and processed becomes a significant problem. Because when the volume of the data exceeds several gigabytes traditional relational databases either do not support such volumes or face performance issues (see [1] for a comparison of the database size limits). Storing and querying very large volumes of data require additional resources; sometimes database clusters are installed for this purpose. However storage and retrieval are not the only problem; the real bottleneck is the ability to analyze the big data volumes and extract useful information such as system faults and diagnostic information.

Additionally in recent years more demanding applications are being developed. Sensors are employed in mission critical applications for real or near-real time intervention. For instance, in some cases it is expected from the sensor applications to detect the system failures before they happen.

Traditional data storage and analysis approaches fail to meet the expectations of new types of sensor application domains where the volume and velocity of the data grow in unprecedented rates. As a result, it becomes necessary to adapt new technologies, namely, big data technologies, to be able to cope with these problems.

This paper outlines the architecture and implementation of a novel, distributed, and scalable sensor data storage and analysis system, based on modern cloud computing and big data technologies. The system uses open source technologies to provide end-to-end sensor data lifecycle management and analysis tools.

2. Background, Related Concepts, and Technologies

2.1. Sensors, Internet of Things, and NoSQL. Sensors are everywhere and the size and variety of the data they produce are growing rapidly. Consecutively, new concepts are emerging as the types and usage of sensors expands steadily.

For example, the statistics shows that amount of the things on the Internet is much larger than the number of the users on the Internet [2]. This inference defines the Internet of things (IoT) as the Internet relating to things. The term "things" on the IoT, first used by Ashton in 1999 [3], is a vision that includes physical objects. These objects, which collect information and send it to the network autonomously, can be RFID tags, sensors, GPS, cameras, and other devices. The connection between IoT and Internet enables the communication between people and objects, objects between themselves, and people between themselves with connections such as Wi-Fi, RFID, GPRS, DSL, LAN, and 3G. These networks produce huge volumes of data, which are difficult to store and analyze with traditional database technologies.

IoT enables interactions among people, objects, and networks via remote sensors. Sensors are devices, which can monitor temperature, humidity, pressure, noise levels, and lighting condition and detect speed, position, and size of an object [4]. Sensor technology has recently become a thriving field including many industrial, healthcare, and consumer applications such as home security systems, industrial process monitoring, medical devices, air-conditioning systems, intelligent washing machines, car airbags, mobile phones, and vehicle tracking systems.

Due to the rapid advances in sensor technologies, the number of sensors and the amount of sensor data have been increasing with incredible rates. Processing and analyzing such big data require enormous computational and storage costs with a traditional SQL database. Therefore the scalability and availability requirements for sensor data storage platform solutions resulted in use of NoSQL databases, which have the ability to efficiently distribute data over many servers and dynamically add new attributes to data records [5].

NoSQL databases, mostly open source, can be divided into following categories.

(i) *Key-Value Stores*. These database systems store values indexed by keys. Examples of this category are *Redis*, Project *Voldemort*, *Riak*, and Tokyo Cabinet.

(ii) *Document Stores*. These database systems store and organize collections of documents, in which each document is assigned a unique key. Examples of this category are *Amazon SimpleDB*, MongoDB, and CouchDB.

(iii) *Wide-Column Stores*. These database systems, also called extensible record stores, store data tables of extensible records that can be partitioned vertically and horizontally across multiple nodes. Examples of this category are HBase, Cassandra, and HyperTable.

Different categories of NoSQL databases, such as key-value, document, and wide-column stores, provide high availability, performance, and scalability for big data. Reference [6] has proposed two-tier architecture with a data model and alternative mobile web mapping solution using NoSQL database CouchDB, which is available on almost all operating systems.

van der Veen et al. [7] have discussed the possibilities to use NoSQL databases such as MongoDB and Cassandra in large-scale sensor network systems. The results show that while Cassandra is the best choice for large critical sensor application, MongoDB is the best choice for a small or medium sized noncritical sensor application. On the other hand, MongoDB has a moderate performance when using virtualization; by contrast, read performance of Cassandra is heavily affected by virtualization.

2.2. Big Data. Using sensors in large quantities results in big volumes of data to be stored and processed. Data is valuable when information within is extracted and used. Information extraction requires tools and algorithms to identify useful information such as fault messages or system diagnostic messages buried deep in the data collected from sensors. Data mining or machine learning can be used for such tasks. However big data analytics requires nontraditional approaches, which are collectively dubbed as big data.

Big data is the name of a collection of theories, algorithms, and frameworks, dealing with the storage and analysis of very large volumes of data. In other words "big data" is a term maturing over time that points a large amount of data which are difficult to store, manage, and analyze using traditional database and software technologies. In recent years, big data analysis has become one of the most popular topics in the IT world and keeps drawing more interest from the academia and the industry alike. The rapid growth in the size, variety, and velocity of data forces developers to build new platforms to manage this extreme size of information. International Data Corporation (IDC) reports that the total amount of data in the digital universe will reach 35 zettabytes by 2020 [8]. IEEE Xplore states that "in 2014, the most popular search terms and downloads in IEEE Xplore were: big data, data mining, cloud computing, internet of things, cyber security, smart grid and next gen wireless (5G)" [9].

Big data has many challenges due to several aspects like variety, volume, velocity, veracity, and value. Variety refers to unstructured data in different forms such as messages, social media conversations, videos, and photos; volume refers to large amounts of data; velocity refers to how fast the data is generated and how fast it needs to be analyzed; veracity refers to the trustworthiness of data; value, the most important V of big data, refers to the worth of the data stored by different organizations [10]. In order to facilitate better understanding of big data challenges described with 5V, Figure 1 shows the different categories to classify big data.

In the light of the categories given in big data classification, big data map can be addressed in seven aspects: (i) data sources, (ii) data type, (iii) content format, (iv) data stores, (v) analysis type, (vi) infrastructure, and (vii) processing framework.

Data sources include the following: (a) human-generated data such as social media data from Facebook and Twitter or text messages, Internet searches, blogs and comments, and personal documents; (b) business transaction data such as banking records, credit cards, commercial transactions, and medical records; (c) machine-generated data from the Internet of things such as home automation systems mobile devices and logs from computer systems; (d) various types

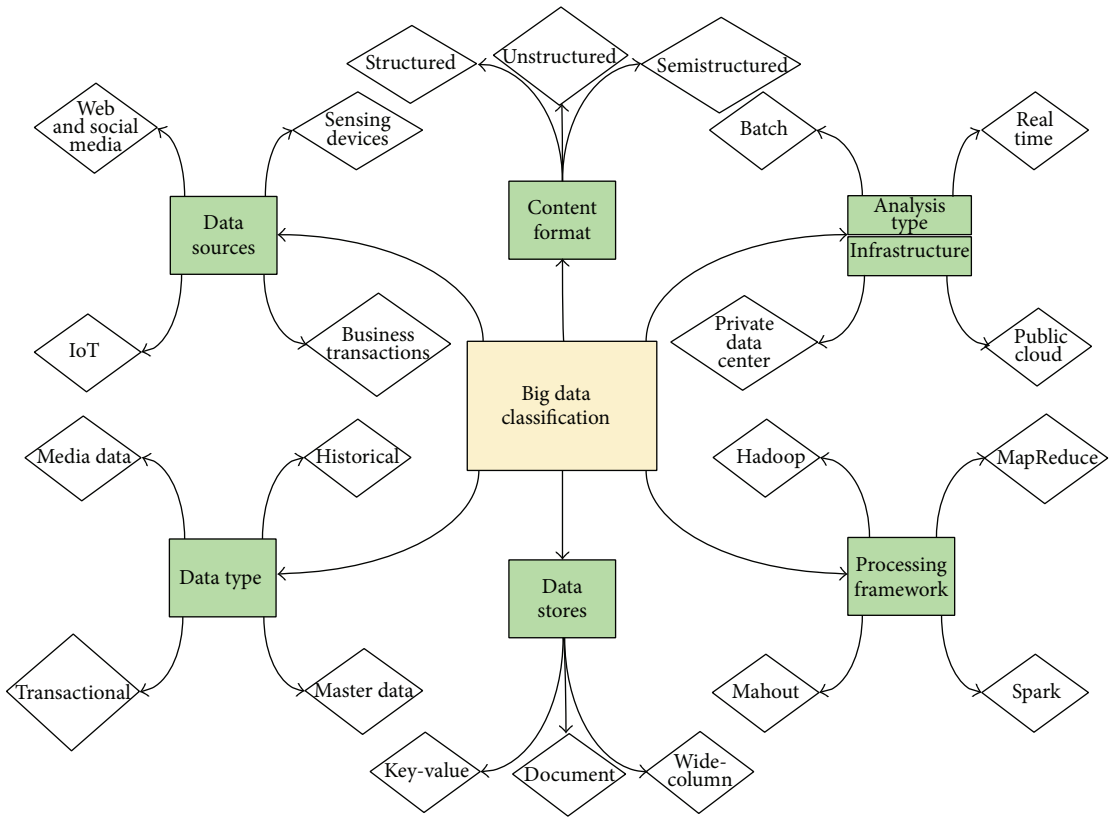

Figure 1: Big data classification (based on [52]).

of sensors such as traffic sensors, humidity sensors, and industrial sensors.

2.3. MapReduce and Hadoop. The amount of data generated from web, sensors, satellites, and many other sources overcomes the traditional data analysis approaches, which pave the way for new types of programming models such as MapReduce. In 2004, Google published the MapReduce paper [11] which demonstrated a new type of distributed programming model that makes it easy to run high-performance parallel programs on big data using commodity hardware. Basically MapReduce programs consist of two major modules, mappers and reducers, which are user-defined programs implemented by using the MapReduce API. Therefore a MapReduce job is composed of several processes such as splitting and distributing the data, mapping and reducing codes, and writing results to the distributed file system. Sometimes analyzing data using MapReduce may require running more than one job. The jobs can be independent of each other or they may be chained for more complex scenarios.

MapReduce paradigm works as shown in Figure 2: MapReduce jobs are controlled by a master node and are splitted into two functions called Map and Reduce. The Map function divides the input data into a group of key-value pairs and the output of each map task is sorted by their key. The Reduce function merges the values into final result.

MapReduce, Google's big data processing paradigm, has been implemented in open source projects like Hadoop [12]. Hadoop has been the most popular MapReduce implementation and is used in many projects from all areas of big data industry [13, 14]. The so-called Hadoop Ecosystem also provides many other big data tools such as Hadoop Distributed File System [15], for storing data on clusters, Pig [16], an engine for parallel data flow execution on Hadoop, HBase [17], Google's Big Table like nonrelational distributed database, Hive [18], a data warehouse software on Hadoop, and data analysis software like Mahout [19].

Major advantages of Hadoop MapReduce framework are scalability, cost effectiveness, flexibility, speed, and resilience to failures [20]. On the other hand, Hadoop does not fully support complex iterative algorithms for machine learning and online processing.

Other MapReduce-like systems are Apache Spark and Shark [21], HaLoop [22], and Twister [23]. These systems provide better support for certain types of iterative statistical and complex algorithms inside a MapReduce-like programming model but still lack most of the data management features of relational database systems [24]. Usually these systems also take advantage of the following: (1) programming languages with functional and parallel capabilities such as Scala, Java, or Python; (2) NoSQL stores; (3) MapReduce-based frameworks [25].

Hadoop uses the Hadoop Distributed File System (HDFS), which is the open source version of Google File System [26]. The data in HDFS is stored on a block-by-block basis. First the files are split into blocks and then are distributed over the Hadoop cluster. Each block in the HDFS

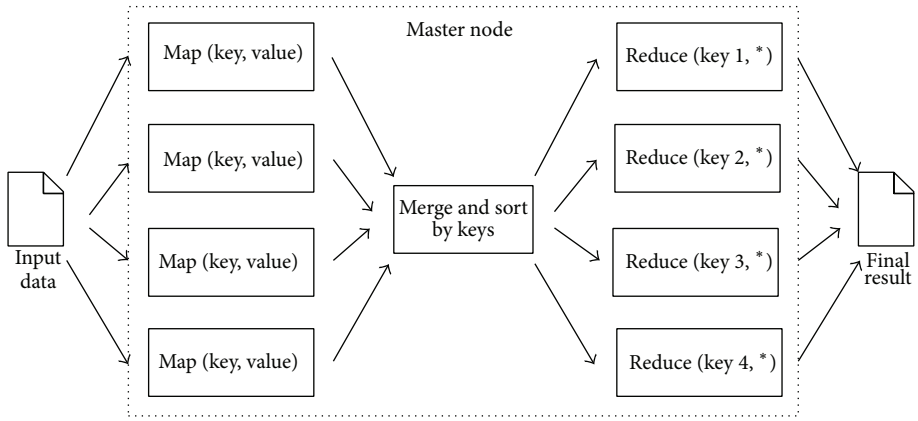

Figure 2: An overview of the Map and Reduce steps.

is 64 MB by default unless the block size is modified by the user [15]. If the file is larger than 64 MB the HDFS splits it from a line where the file size does not exceed the maximum block size and the rest of the lines (for text input) are moved to a new block.

Hadoop uses master-slave architecture. Name Node and Job Tracker are master nodes whereas Data Node and Task Tracker are slave nodes in the cluster. The input data is partitioned into blocks and these blocks are placed into Name Node which holds the metadata of the blocks so the Hadoop system knows which block is stored on which Data Node. And if one node fails it does not spoil the completion of the job because Hadoop knows where the replicas of those blocks are stored [27]. Job Tracker and Task Tracker track the execution of the processes. They have a similar relation with Name Node and Data Node. Task Tracker is responsible for running the tasks and sending messages to Job Tracker. Job Tracker communicates with Task Tracker and keeps record of the running processes. If Job Tracker detects that a Task Tracker is failed or is unable to complete its part of the job, it schedules the missing executions on another Task Tracker [14].

2.4. Cloud Computing. Running Hadoop efficiently for big data requires clusters to be set up. Advances in the virtualization technology have significantly reduced the cost of setting up such clusters; however they still require major economic investments, license fees, and human intervention in most cases. Cloud computing offers a cost-effective way of providing facilities for computation and for processing of big data and also serves as a service model to support big data technologies.

Several open source cloud computing frameworks such as OpenStack [28], OpenNebula [29], Eucalyptus [30], and Apache CloudStack allow us to set up and run infrastructure as a service (IaaS-cloud model). We can set up platforms as a service (PaaS) such as Hadoop on top of this infrastructure for big data processing.

Hadoop cluster can be set up by installing and configuring necessary files on the servers. However it can be a daunting and challenging work when there are hundreds or even thousands of servers to be used as Hadoop nodes

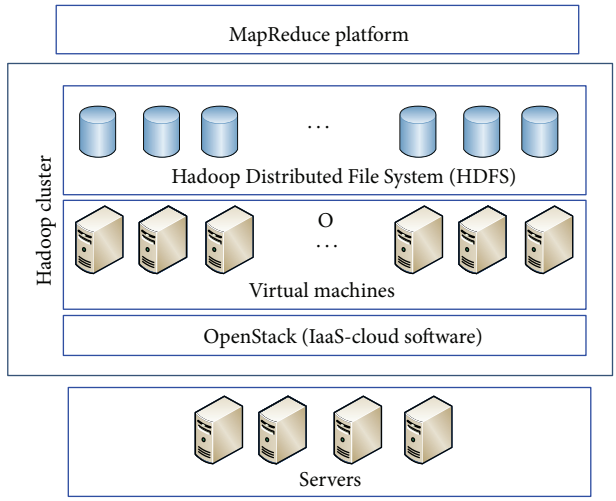

Figure 3: OpenStack Cloud + Hadoop integration and architecture.

in a cluster. Cloud systems provide infrastructure, which is easy to scale and easy to manage the network and the storage and provides fault tolerance features. Gunarathne et al. [31] show the advantages and challenges of running MapReduce in cloud environments. They state that although cloud computing provides storage and other services which meets the distributed computing framework needs, it is less reliable than "their traditional cluster counterparts and do not provide the high-speed interconnects needed by frameworks such as MPI" [31].

The Hadoop platform created for this study is shown in Figure 3.

There are several options for setting up a Hadoop cluster. Paid cloud systems like Amazon EC2 provide EMR [32] clusters for running MapReduce jobs. In EC2 cloud the input data can be distributed to Hadoop nodes through uploading files over the master node. Because pricing in the clouds is on a pay as go basis, customers do not have to pay for the idle nodes. Amazon shuts down the rented instances after the job completes. In this case, all the data will be removed from the system. For example, if the user wants to run another job over the preused data he/she has to upload it again. If data is

stored on Amazon Simple Storage Service (Amazon S3) [32] users can use it as long as he/she pays for the storage. Amazon also provides some facilities for monitoring working Hadoop jobs as well.

2.5. Big Data Analysis. Analyzing big data requires use of data-mining or machine-learning algorithms. There are many user-friendly machine-learning frameworks such as RapidMiner [33] and Weka [34]. However, these traditional frameworks do not scale to big data due to their memory constraints. Several open source big data projects have implemented many of these algorithms. One of these frameworks is Mahout [19], which is a distributed machine-learning framework and licensed under the Apache Software Foundation License.

Mahout provides various algorithms ranging from classification to collaborative filtering and clustering, which can be run in parallel on clusters. The goal of Mahout is basically to build a scalable machine-learning library to be used on Hadoop [35]. As such, the whole task for analysis of large datasets can be divided into a set of many subtasks and the result is the combination of the results from all of the subtasks.

Ericson and Palickara compared the performance of various classification and clustering algorithms using Mahout library on two different processing systems: Hadoop and Granules [36]. Their results showed that the processing time of Granules implementation is faster than Hadoop, which spends the majority of the processing time to load the state from file on every step, for k-means, fuzzy k-means, Dirichlet, and LDA (latent Dirichlet allocation) clustering algorithms. They saw the increased standard deviation for both Naïve Bayes and Complementary Bayes classification algorithms in Granules implementation. Esteves et al. [37] evaluated the performance of k-means clustering algorithm on Mahout using a large dataset. The tests were run on Amazon EC2 instances, demonstrating that the execution times or clustering times of Mahout decrease, as the number of node increases and the gain in performance reaches from 6% to 351% when the data file size is increased from 66 MB to 1.1 GB. As a result, Mahout demonstrates bad performance and no gain for files smaller than 128 MB. Another study described by [37] presented a performance analysis of two different clustering algorithms: k-means and mean shift using Mahout framework. The experimental results have shown that k-means algorithm has better performance than mean shift algorithm, if size of the files is over 50%.

MLLib [38], a module of Spark [21], an in-memory-based distributed machine-learning framework developed at the Berkeley AMPLab, is also licensed under the Apache Software License like Mahout. It is a fast and flexible iterative computing framework, which aims to create and analyze large-scale data hosted in memory. It also provides high-level APIs in Java, Python, and Scala for working with distributed data similar to Hadoop and presents an in-memory processing solution offered for Hadoop. Spark supports running in four cluster modes as follows:

(i) standalone deploy mode, which enables Spark to run on a private cluster using a set of deploy scripts; additionally all Spark processes are run in the same Java virtual machine (JVM) process in standalone local mode;

(ii) Amazon EC2, which enables users to launch and manage Spark clusters on;

(iii) Apache Mesos, which dynamically provides sharing the resources between Spark and other frameworks;

(iv) Hadoop YARN which is commonly referred to as Hadoop 2, which allows Spark drivers to run in the application master.

When machine-learning algorithms are performed on distributed frameworks using MapReduce two approaches are possible: all iteration results can be written to the disk and read from the disk (Mahout) and all iteration results can be stored in memory (Spark). The fact that processing data from memory will be inherently faster than from disk, Spark provides significant performance gain when compared to Mahout/Hadoop.

Spark presents a new distributed memory abstraction, called resilient distributed datasets (RDDs), which provides a data structure for in-memory computations on large clusters. RDDs can achieve fault tolerance, meaning that if a given task fails due to some reasons such as hardware failures and erroneous user code, lost data can be recovered and reconstructed automatically on the remaining tasks [39]. Spark is more powerful and useful for iterative computations than existing cluster computing frameworks, by using data abstraction for programming including RDDs, broadcast variables, and accumulators [21]. With recent releases of Spark, many rich tools such as a database (Spark SQL instead of Shark SQL), a machine-learning library (MLLib), and a graph engine (GraphX) have also been released. MLLib [38] is a Spark component to implement machine-learning algorithms, including classification, clustering, linear regression, collaborative filtering, and decomposition. Due to rapid improvement of Spark, MLLib has lately attracted more attention and is supported by developers from open source community.

The comparison results of Spark and Hadoop performances presented by [40] show that Spark outperforms Hadoop when executing simple programs such as WordCount and Grep. In another similar study [41], it has been shown that k-means algorithm on Spark runs about 5 times faster than that on MapReduce; even the size of data is very small. On the contrary, if dataset consistently varies during the process, Spark loses the advantage over MapReduce. Lawson [42] proposed a distributed method named alternating direction method of multipliers (ADMM) to solve optimization problems using Apache Spark. The result of another study [43], which preferred to implement the proposed distributed method on Spark instead of MapReduce due to the inefficiency on iterative algorithms, demonstrated that the distributed Newton method was efficient for training logistic regression and linear support vector machine with fault tolerance provided by Spark. The performance comparisons of Hadoop, Spark, and DataMPI using k-means and Naïve Bayes benchmarks as the workloads are described

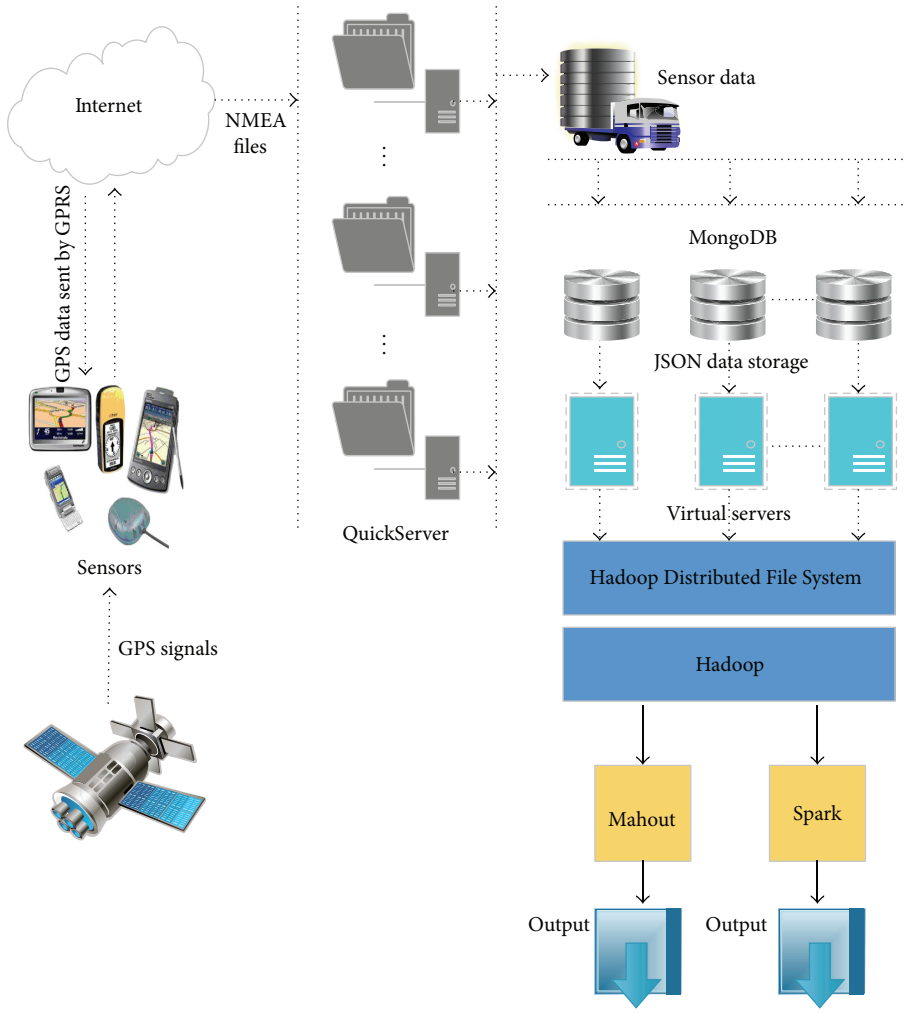

Figure 4: System architecture.

in [44]. The results show that DataMPI and Spark can use CPU more efficiently than Hadoop with 39% and 41% ratios, respectively. Several similar studies as well point to the fact that Spark is well suited for iterative computations and has other advantages for scalable machine-learning applications, when compared to distributed machine-learning frameworks based on MapReduce paradigm.

3. System Architecture

We have created an end-to-end sensor data lifecycle management and analysis system using the aforementioned technologies. The system uses open source software and provides a distributed and scalable infrastructure for supporting as many sensors as needed.

The overview of the proposed system is illustrated in Figure 4. The system architecture consists of three main parts: (1) data harvesting subsystem, (2) data storage subsystem, and (3) data analysis subsystem. The application platform used in the system is Sun Fire X4450 servers with 24 processing cores of Intel 3.16 GHz CPU and 64 GB of memory, using Ubuntu 14.04 as the host operating system.

In this study we used GPS sensors as data generators; however the system architecture is appropriate for other types of sensor networks since the data harvesting subsystem can collect any type of sensor data published through TCP or UDP channels.

3.1. Sensor Data Harvesting Subsystem. GPS is one of the most commonly used technologies for location detection, which is a space-based satellite navigation system for providing time and location information of the receivers globally [45]. It became fully operational in 1995 and since then has been used in numerous industrial and academic projects.

One major use of GPS is vehicle tracking applications. In this study we use a commercial vehicle tracking system called Naviskop [46], developed in Firat Technopark, Elazig, Turkey. Naviskop has been in use for almost a year and the authors have active collaboration in the development of the system. We used GPS sensors mounted on 45 different vehicles. The identity of the drivers and vehicles is not used in the study.

GPS sensors are mostly used in tracking the location of the objects in real time as well as for checking the past location history. However in most of the GPS applications

data are not analyzed afterwards. In this study we use the location data from the vehicles for discovering hidden, interesting information. For example, by applying machine-learning algorithms, GPS data can reveal the driving habits of individuals, most popular places which people visit with their vehicles, and traffic density for a certain period of the day. Several academic studies have investigated the use of location data with data-mining and machine-learning algorithms [47–50].

GPS receivers mounted on the vehicles have the ability to report their location via GPRS. The sensors open a connection to the TCP server in several situations such as in every 100 m location change or in every 30 degrees of turns.

We use QuickServer, an open source Java library for quick creation of robust and multithreaded, multiclient TCP server applications and powerful server applications [51]. QuickServer supports multiclient TCP server applications and secure connections like SSL and TLS, thread per client, nonblocking communications, and so forth. It has a remote administration interface called QSAdminServer which can be used to manage every aspect of the server software.

QuickServer is used to collect the real time data sent by the GPS servers. We created a data filtering and parsing program on the server for immediately extracting useful information and inserting it into the database.

3.2. Sensor Data Storage Subsystem. Data collected from the sensors are usually stored in some sort of a data storage solution. However as the number of sensors and hence the amount of data increase it becomes a nontrivial task to continuously store it. Traditional sensor data storage solutions advise storing data for only certain period of times. However the data collected from the sensors are valuable since they might carry hidden motifs for faults or diagnostic information. For this reason we have created a scalable, distributed data storage subsystem for storing sensor data until they are analyzed.

Open source NoSQL databases provide efficient alternatives for large amount of sensor data storage. In this study we used MongoDB, a popular open source NoSQL database [53]. MongoDB is a document-oriented database with support for storing JSON-style documents. It provides high performance, high availability, and easy scalability. Documents stored in MongoDB can be mapped to programming language data types. Dynamic schema support makes polymorphism easy to implement. MongoDB servers can be replicated with automatic master failover. To scale the databases, automatic clustering (sharding) distributes data collections across machines.

MongoDB has been investigated in several studies and been used in various types of commercial and academic projects [54–58].

The main reason for using MongoDB in our implementation is providing high-performance write support for QuickServer. It also allows us to easily scale the databases for cases where large numbers of sensors are used.

3.3. Sensor Data Analysis Subsystem. Storing sensor data indefinitely is a very important feature for the system. However sensor data must be analyzed to find important information such as early warning messages and fault messages. Data analysis can be done by simply using statistical methods as well as by using more complex data-mining or machine-learning algorithms. In this study we have created a scalable, distributed data analysis subsystem using big data technologies. Our goal is to be able to run advanced machine-learning algorithms on the sensor data for finding valuable information.

Big data processing requires processing power as well as storage support usually provided by computing clusters. Clusters are traditionally created using multiple servers; however virtualization allows us to maximize the resource utilization and decrease the cluster creation costs. Virtualization helps us in running several operating systems on a single physical machine which in turn can be used as cluster nodes. On the other hand, since most virtualization software requires high license fees or extensive professional background, we utilize open source cloud computing software called OpenStack for creating the compute nodes for the Hadoop cluster.

OpenStack is the popular technology cloud computing that offers many opportunities for big data processing with scalable computational clusters and advanced data storage systems for applications and science researchers [28, 59–61]. Cloud computing stack can be categorized in three service models: infrastructure as a service (IaaS), platform as a service (PaaS), and software as a service (SaaS) where IaaS is most flexible and basic cloud computing model. IaaS provides the access and management to computing hardware, storage, networking, and operating systems with a configurable virtual server [62]. IaaS providers include Amazon EC2, Rackspace Cloud, and Google Compute Engine (GCE). OpenStack, as used in this study, is an IaaS-cloud computing software project based on the code developed by Rackspace and NASA. OpenStack offers a scalable, flexible, and open source cloud computing management platform. The comparative study in [60] shows that OpenStack is the best reference solution of open source cloud computing. OpenStack provides a web based GUI for management of the system and creating/deleting VMs. Figure 5 shows the overview of the resource usage in our OpenStack installation.

In this study, we created a private cloud using OpenStack and run 6 instances of virtual machines (master node operates as a worker too) as Hadoop cluster nodes (see Figure 6).

4. Sensor Data Analysis Results

To analyze data on the aforementioned architecture we use distributed machine-learning algorithms. Apache Mahout and MLLib by Apache Spark are open source distributed frameworks for big data analysis. We use both frameworks for implementing clustering analysis on the GPS sensor data. The clustering results might be used for road planning or interpreted to find most crowded places in the cities or most popular visitor destinations, traffic density in certain time periods, and so forth. We map data stored in MongoDB to HDFS running on the cluster nodes.

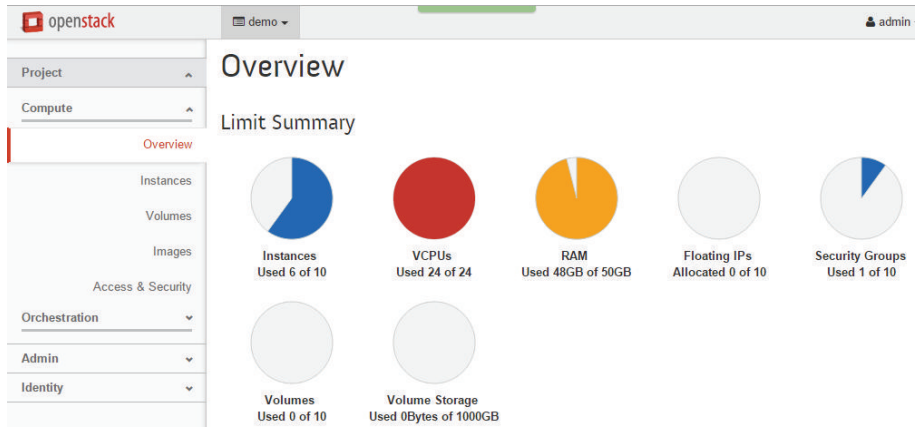

Figure 5: OpenStack overview screen.

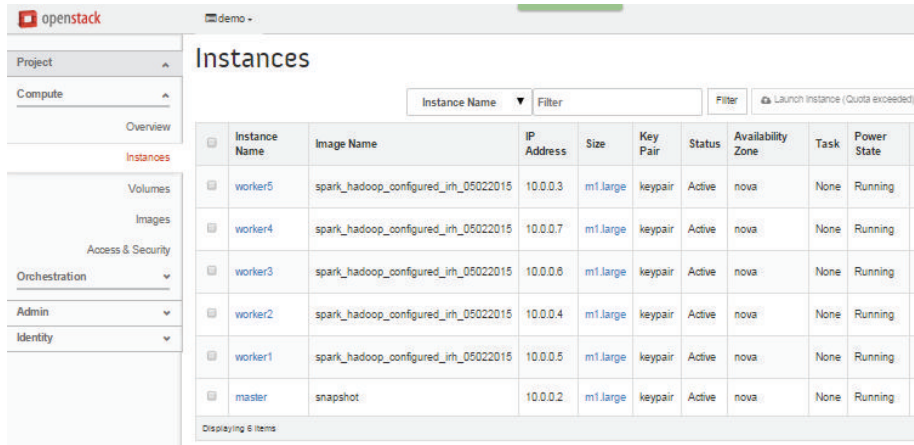

Figure 6: OpenStack GUI screenshot shows the cluster node specifications.

GPS sensors provide us with several important pieces of information such as the latitude, longitude, and altitude of the object being tracked, time, and ground speed. These measurements can be used for various purposes. In this study we used latitude and longitude data from vehicle GPS sensors.

Several studies demonstrate usage of machine-learning and data-mining algorithms on spatial data [63–66]. However the size of data is a significant limitation for running these algorithms since most of the algorithms are computationally complex and require high amount of resources. Big data technologies can be used to analyze very large spatial datasets.

We have used k-means algorithm for clustering two-dimensional GPS position data. k-means algorithm is a very popular unsupervised learning algorithm. It aims to assign objects to groups. All of the objects to be grouped need to be represented as numerical features. The technique iteratively assigns n points to k clusters using distance as a similarity factor until there is no change in which point belongs to which cluster.

k-means clustering has been applied to spatial data in several studies. Reference [67] describes clustering rice crop statistics data taken from the Agricultural Statistics of India. However spatial data clustering using k-means becomes impossible on low end computers as the number of points exceeds several millions.

In this study we use our architecture to cluster large datasets with millions of points. Performance results shown in Figures 7, 8, and 9 show that the system is able to cluster very large numbers of points efficiently.

Table 1 shows the data file sizes used in the tests.

As a reference, we first run Weka on a desktop machine with 8 GB of RAM and Intel i5-3470 CPU. Table 2 and Figure 7 show the results. Weka [68] is a well-known data-mining and machine-learning software and has been used in many studies.

As Table 2 shows Weka demonstrates good performance for data with a relatively small number of coordinates. However as the number of points increases the performance of Weka decreases and for over 2 million points it gives out memory exceptions. By changing Java heap size, this limit can be increased, but there will always be an upper limit depending on the computer specifications.

Table 3 shows the execution times for k-means clustering on our system using Spark for up to 5 million coordinates. As the table shows the system demonstrates worse performance

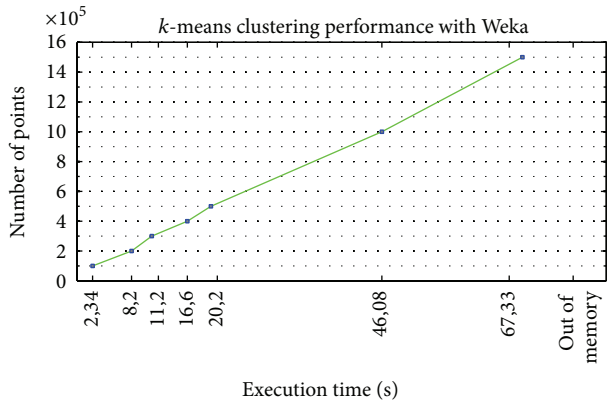

Figure 7: k-means performance with Weka.

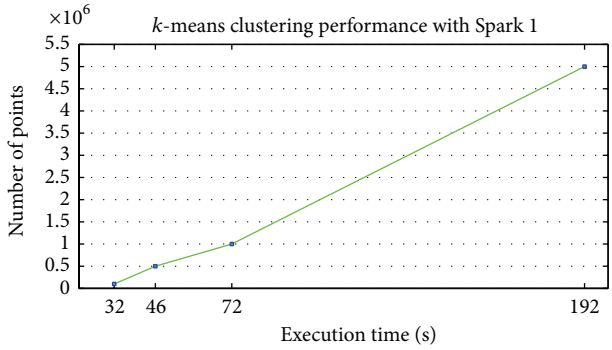

Figure 8: k-means performance with Spark, up to 5 million points.

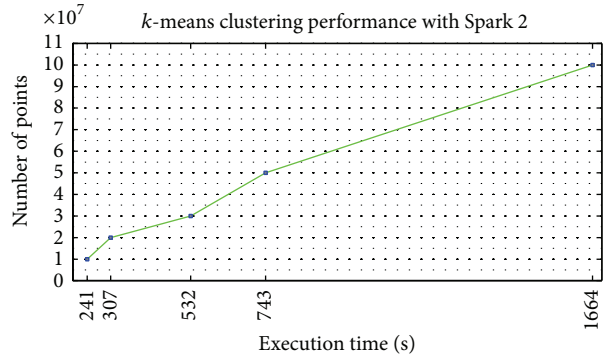

Figure 9: k-means performance with Spark, up to 100 million points.

for a small number of points but it can process 5 million points in a reasonable time.

However the real advantage of using distributed algorithms can be seen in Table 4, where performance results of Spark k-means clustering are shown for a very large number of points.

As Figure 9 shows the execution time on the Spark cluster increases linearly and the system can analyze millions of coordinates without any performance issues.

Table 1: Input data sizes.

Number of points (millions)	File size
1	14 MB
10	134 MB
20	268 MB
30	401 MB
50	668 MB
100	1.4 GB

Table 2: Weka k-means clustering performance results.

Number of points	Execution time (sec)
100.000	2,34
200.000	8,23
300.000	11,29
400.000	16,67
500.000	20,23
1.000.000	46,08
1.500.000	67,33
2.000.000	Out of memory
3.000.000	Out of memory

Table 3: k-means clustering performance results with Spark 1.

Number of points	Execution time (sec)
100.000	32
500.000	46
1.000.000	72
5.000.000	192

Table 4: k-means clustering performance results with Spark 2.

Number of points	Execution time (sec)
10.000.000	241
20.000.000	307
30.000.000	532
50.000.000	743
100.000.000	1664

5. Conclusion

In this paper we demonstrated the architecture and test results for a distributed sensor data collection, storage, and analysis system. The architecture can be scaled to support a large number of sensors and big data sizes. It can be used to support geographically distributed sensors and collect sensor data via a high-performance server. The test results show that the system can execute computationally complex data analysis algorithms and shows high performances with big sensor data. As a result we show that, using open source technologies, modern cloud computing and big data frameworks can be utilized for large-scale sensor data analysis requirements.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

References

[1] Comparison of Relational Database Systems, http://en.wikipedia.org/wiki/Comparison_of_relational_database_management_system.

[2] G. Press, Internet of Things by the Numbers: Market Estimates and Forecasts, http://www.forbes.com/.

[3] K. Ashton, That "Internet of Things" Thing, 2015, http://www.rfidjournal.com/articles/view?4986.

[4] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci, "Wireless sensor networks: a survey," *Computer Networks*, vol. 38, no. 4, pp. 393–422, 2002.

[5] R. Cattell, "Scalable SQL and NoSQL data stores," *ACM SIGMOD Record*, vol. 39, no. 4, pp. 12–27, 2010.

[6] M. Miler, D. Medak, and D. Odobašić, "Two-tier architecture for web mapping with NoSQL database couch DB," *Geospatial Crossroads GI Forum*, vol. 11, pp. 62–71, 2011.

[7] J. S. van der Veen, B. van der Waaij, and R. J. Meijer, "Sensor data storage performance: SQL or NoSQL, physical or virtual," in *Proceedings of the IEEE 5th International Conference on Cloud Computing (CLOUD '12)*, pp. 431–438, IEEE, June 2012.

[8] J. Gantz and D. Reinsel, *Extracting Value from Chaos State of the Universe*, IDC (International Data Corporation), 2011.

[9] IEEE XPLORE, "Year in Review: Top Search Terms in IEEE Xplore," http://ieeexplore.ieee.org/Xplore/.

[10] A. Katal, M. Wazid, and R. H. Goudar, "Big data: issues, challenges, tools and good practices," in *Proceedings of the 6th International Conference on Contemporary Computing (IC3 '13)*, pp. 404–409, IEEE, Noida, India, August 2013.

[11] J. Dean and S. Ghemawat, "MapReduce: simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[12] Official Hadoop Web Site, 2015, http://hadoop.apache.org/.

[13] C. Sweeney, L. Liu, S. Arietta, J. Lawrence, and B. S. Thesis, *HIPI: a Hadoop Image Processing Interface for Image-Based Mapreduce Tasks*, University of Virginia, Charlottesville, Va, USA, 2011.

[14] T. White, *Hadoop: The Definitive Guide*, O'Reilly Media, 2009.

[15] D. Borthakur, *HDFS Architecture Guide*, Hadoop Apache Project, 2008.

[16] C. Olston, B. Reed, U. Srivastava, R. Kumar, and A. Tomkins, "Pig latin: a not-so-foreign language for data processing," in *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD '08)*, pp. 1099–1110, ACM, June 2008.

[17] L. George, *HBase: The Definitive Guide*, O'Reilly Media, 2011.

[18] A. Thusoo, J. S. Sarma, N. Jain et al., "Hive: a warehousing solution over a map-reduce framework," *Proceedings of the VLDB Endowment*, vol. 2, no. 2, pp. 1626–1629, 2009.

[19] S. Owen, R. Anil, T. Dunning, and E. Friedman, *Mahout in Action*, Manning Publications, 2011.

[20] M. Nemschoff, Big Data: 5 Major Advantages of Hadoop, http://www.itproportal.com/.

[21] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark: cluster computing with working set," in *Proceedings of the 2nd USENIX Conference on Hot Topics in Cloud Computing*, 2010.

[22] Y. Bu, B. Howe, M. Balazinska, and M. D. Ernst, "HaLoop: efficient iterative data processing on large clusters," *Proceedings of the VLDB Endowment*, vol. 3, no. 1-2, pp. 285–296, 2010.

[23] J. Ekanayake, H. Li, B. Zhang et al., "Twister: a runtime for iterative mapreduce," in *Proceedings of the ACM International Symposium on High Performance Distributed Computing (HPDC '10)*, pp. 810–818, ACM, June 2010.

[24] S. Madden, "From databases to big data," *IEEE Internet Computing*, vol. 16, no. 3, pp. 4–6, 2012.

[25] D. Kourtesis, J. M. Alvarez-Rodríguez, and I. Paraskakis, "Semantic-based QoS management in cloud systems: current status and future challenges," *Future Generation Computer Systems*, vol. 32, no. 1, pp. 307–323, 2014.

[26] S. Ghemawat, H. Gobioff, and S. T. Leung, "The google file system," in *Proceedings of the 19th ACM Symposium on Operating Systems Principles (SOSP '03)*, pp. 29–43, October 2003.

[27] A. Bialecki, M. Cafarella, D. Cutting, and O. O'Malley, *Hadoop: A Framework for Running Applications on Large Clusters Built of Commodity Hardware*, Wiki, 2005, http://lucene.apache.org/hadoop.

[28] OpenStack, 2015, http://www.openstack.org.

[29] OpenNebula Web page, 2015, http://www.opennebula.org.

[30] Eucalyptus, 2015, https://www.eucalyptus.com/eucalyptus-cloud/iaas.

[31] T. Gunarathne, T.-L. Wu, J. Qiu, and G. Fox, "MapReduce in the clouds for science," in *Proceedings of the 2nd IEEE International Conference on Cloud Computing Technology and Science (CloudCom '10)*, pp. 565–572, IEEE, December 2010.

[32] Amazon Web Services, 2015, http://aws.amazon.com.

[33] RapidMiner Predictive Analysis, 2015, https://rapidminer.com/.

[34] G. Holmes, A. Donkin, and I. H. Witten, "Weka: a machine learning workbench," in *Proceedings of the 2nd Australian and New Zealand Conference on Intelligent Information Systems*, pp. 357–361, Brisbane, Australia, December 1994.

[35] A. Mahout, "Scalable machine-learning and data-mining library," http://mahout.apache.org/.

[36] K. Ericson and S. Pallickara, "On the performance of high dimensional data clustering and classification algorithms," *Future Generation Computer Systems*, vol. 29, no. 4, pp. 1024–1034, 2013.

[37] R. M. Esteves, R. Pais, and C. Rong, "K-means clustering in the cloud—a Mahout test," in *Proceedings of the IEEE Workshops of International Conference on Advanced Information Networking and Applications (WAINA '11)*, pp. 514–519, IEEE, 2011.

[38] Spark MLLib scalable machine learning library, https://spark.apache.org/mllib/.

[39] M. Zaharia, M. Chowdhury, T. Das et al., "Resilient distributed datasets: a fault-tolerant abstraction for in-memory cluster computing," in *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation (NSDI '12)*, USENIX Association, 2012.

[40] S. Shahrivari, "Beyond batch processing: towards real-time and streaming big data," *Computers*, vol. 3, no. 4, pp. 117–129, 2014.

[41] H. Wang, B. Wu, S. Yang, and B. Wang, "Research of decision tree on YARN using 16 MapReduce and spark," in *Proceedings of the The 2014 World Congress in Computer Science, Computer Engineering, and Applied Computing*, Las Vegas, Nev, USA, 2014.

[42] D. Lawson, *Alternating Direction Method of Multipliers Implementation Using Apache Spark*, 2014.

[43] C.-Y. Lin, C.-H. Tsai, C.-P. Lee, and C.-J. Lin, "Large-scale logistic regression and linear support vector machines using spark," in *Proceedings of the IEEE International Conference on Big Data*, pp. 519–528, Washington, DC, USA, October 2014.

[44] F. Liang, C. Feng, X. Lu, and Z. Xu, "Performance benefits of DataMPI: a case study with BigDataBench," in *Big Data Benchmarks, Performance Optimization, and Emerging Hardware*, vol. 8807 of *Lecture Notes in Computer Science*, pp. 111–123, Springer International Publishing, Cham, Switzerland, 2014.

[45] Wikipedia, "Global Positioning System," http://en.wikipedia.org/wiki/Global_Positioning_System.

[46] Yonca CBS, "Naviskop Vehicle Tracking Systems," 2015, http://www.naviskop.com/.

[47] J. Han, K. Koperski, and N. Stefanovic, "GeoMiner: a system prototype for spatial data mining," *ACM SIGMOD Record*, vol. 26, no. 2, pp. 553–556, 1997.

[48] C. J. Moran and E. N. Bui, "Spatial data mining for enhanced soil map modelling," *International Journal of Geographical Information Science*, vol. 16, no. 6, pp. 533–549, 2002.

[49] R. T. Ng and J. Han, "Clarans: a method for clustering objects for spatial data mining," *IEEE Transactions on Knowledge and Data Engineering*, vol. 14, no. 5, pp. 1003–1016, 2002.

[50] S. Shekhar, P. Zhang, and Y. Huang, "Spatial data mining," in *Data Mining and Knowledge Discovery Handbook*, pp. 833–851, Springer, 2005.

[51] Quick Server, February 2015, http://www.quickserver.org/.

[52] S. K. Divakar Mysore and S. Jain, *Big Data Architecture and Patterns, Part 1: Introduction to Big Data Classification and Architecture*, IBM Big Data and Analytics, Technical Library, 2013.

[53] P. Membrey, E. Plugge, and D. Hawkins, *The Definitive Guide to MongoDB: the noSQL Database for Cloud and Desktop Computing*, Apress, 2010.

[54] A. Boicea, F. Radulescu, and L. I. Agapin, "MongoDB vs oracle-database comparison," in *Proceedings of the 3rd International Conference on Emerging Intelligent Data and Web Technologies (EIDWT '12)*, pp. 330–335, September 2012.

[55] E. Dede, M. Govindaraju, D. Gunter, R. S. Canon, and L. Ramakrishnan, "Performance evaluation of a MongoDB and Hadoop platform for scientific data analysis," in *Proceedings of the 4th ACM Workshop on Scientific Cloud Computing (ScienceCloud '13)*, pp. 13–20, ACM, June 2013.

[56] Y. Liu, Y. Wang, and Y. Jin, "Research on the improvement of MongoDB Auto-Sharding in cloud environment," in *Proceedings of the 7th International Conference on Computer Science & Education (ICCSE '12)*, pp. 851–854, IEEE, Melbourne, Australia, July 2012.

[57] Z. Parker, S. Poe, and S. V. Vrbsky, "Comparing nosql mongodb to an sql db," in *Proceedings of the 51st ACM Southeast Conference*, ACM, April 2013.

[58] Z. Wei-Ping, L. Ming-Xin, and C. Huan, "Using MongoDB to implement textbook management system instead of MySQL," in *Proceedings of the IEEE 3rd International Conference on Communication Software and Networks (ICCSN '11)*, pp. 303–305, IEEE, May 2011.

[59] K. Jackson, *OpenStack Cloud Computing Cookbook*, Packt Publishing Ltd., 2012.

[60] O. Sefraoui, M. Aissaoui, and M. Eleuldj, "OpenStack: toward an open-source solution for cloud computing," *International Journal of Computer Applications*, vol. 55, no. 3, pp. 38–42, 2012.

[61] C. P. Chen and C.-Y. Zhang, "Data-intensive applications, challenges, techniques and technologies: a survey on Big Data," *Information Sciences*, vol. 275, pp. 314–347, 2014.

[62] S. Gao, L. Li, W. Li, K. Janowicz, and Y. Zhang, "Constructing gazetteers from volunteered Big Geo-Data based on Hadoop," *Computers, Environment and Urban Systems*, 2014.

[63] S. Brooker, S. Clarke, J. K. Njagi et al., "Spatial clustering of malaria and associated risk factors during an epidemic in a highland area of western Kenya," *Tropical Medicine and International Health*, vol. 9, no. 7, pp. 757–766, 2004.

[64] T. Cheng, J. Haworth, B. Anbaroglu, G. Tanaksaranond, and J. Wang, "Spatiotemporal data mining," in *Handbook of Regional Science*, pp. 1173–1193, Springer, Berlin, Germany, 2014.

[65] S. Wang and H. Yuan, "Spatial data mining: a perspective of big data," *International Journal of Data Warehousing and Mining*, vol. 10, no. 4, pp. 50–70, 2014.

[66] Y. J. Akhila, A. Naik, B. Hegde, P. Shetty, and A. J. K. Mohan, "SD miner—a spatial data mining system," *International Journal of Research*, vol. 1, no. 5, pp. 563–567, 2014.

[67] R. Sharma, M. A. Alam, and A. Rani, "K-means clustering in spatial data mining using weka interface," *International Journal of Computer Applications*, pp. 26–30, 2012, Proceedings of the International Conference on Advances in Communication and Computing Technologies (ICACACT '12).

[68] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, "The WEKA data mining software: an update," *ACM SIGKDD Explorations Newsletter*, vol. 11, no. 1, pp. 10–18, 2009.