Sādhanā (2019)44:237
https://doi.org/10.1007/s12046-019-1220-z

Real-time BigData and Predictive Analytical Architecture for healthcare application

VIKAS CHAUHAN[1,*], RAGHVENDRA GAUR[2], ARUNA TIWARI[1] and ANUPAM SHUKLA[2]

[1]Department of Computer Science and Engineering, Indian Institute of Technology Indore, Indore, India
[2]Indian Institute of Information Technology and Management, Gwalior, India
e-mail: phd1701101006@iiti.ac.in; raghav.raghvendragaur@gmail.com; artiwari@iiti.ac.in; anupam1165@gmail.com

MS received 13 April 2017; revised 12 July 2019; accepted 26 August 2019

Abstract. The amount of data produced within health informatics has grown to be quite vast. The large volume of data generated by various vital sign monitoring devices needs to be analysed in real time to alert the care providers about changes in a patients condition. Data processing in real time has complex challenges for the large volume of data. The real-time system should be able to collect millions of events per seconds and handle parallel processing to extract meaningful information efficiently. In our study, we have proposed a real-time BigData and Predictive Analytical Architecture for healthcare application. The proposed architecture comprises three phases: (1) collection of data, (2) offline data management and prediction model building and (3) real-time processing and actual prediction. We have used Apache Kafka, Apache Sqoop, Hadoop, MapReduce, Storm and logistic regression to predict an emergency condition. The proposed architecture can perform early detection of emergency in real time, and can analyse structured and unstructured data like Electronic Health Record (EHR) to perform offline analysis to predict patient's risk for disease or readmission. We have evaluated prediction performance on different benchmark datasets to detect an emergency condition of any patient in real time and possibility of readmission.

Keywords. Bigdata; Hadoop; real-time streaming; Storm; Kafka; regression; healthcare.

1. Introduction

BigData refers to the unprocessed, unstructured and complex data in volumes on the range of exabytes (10^{18}) and beyond processed at high velocity. BigData is identified by volume, variety, velocity, storage, processing and management of data. In such volumes, our traditional approach such as relational database systems is not only tedious but also very costly. Such volume exceeds the capacity of current traditional storage and processing systems [1]. Healthcare devices generate a large amount of data. In healthcare, BigData generally refers to electronically generated health records or datasets. These health records include clinical data from written notes and prescriptions by physicians, medical images (like X-rays, sonography, etc.), laboratory reports, pharmaceutical reports, insurance records and other data. Similarly, a large volume of data is generated by monitoring devices from hospitals and in-home devices. These generated data are so large and complex that it is nearly impossible for a traditional system to store, manipulate and manage these data in an efficient manner.

Small physician clinics to a large group of hospitals and large healthcare organisations effectively use BigData. Predictive modelling in BigData can be used for more targeted research and development pipeline in drugs and devices, which will be leaner and faster, and can result in lower attrition as well. Structured and unstructured data like Electronic Health Record (EHR), medical and operational data, and genomic data jointly analysed to match outcomes of treatment can predict risk for diseases or risk for readmission and can help in providing adequate care. Similarly, data obtained from various monitoring devices can be captured and analysed further for safety monitoring and adverse event prediction. Modern healthcare facilities are looking to provide more proactive care to their patients by constantly monitoring vital signs. New sensor technology has made it possible to monitor essential signs anytime–anywhere. The data from various vital sign monitors need to be stored and analysed in real time to alert the care providers, so they know instantly about changes in patients. Also, by continuously monitoring vital signs of patients, the condition of emergency can be predicted. For example, if we could look back at each patient's vitals for the 3 h preceding a heart attack, we would predict when the other patient might have a heart attack. Data processing in real time is a very

*For correspondence

Published online: 19 November 2019

challenging and complex task. Generally, BigData is categorised into volume, velocity and variety of data. Managing this velocity and variety of data is not an easy task. In our study, we have proposed a real-time BigData and Predictive Analytical Architecture for healthcare application.

The proposed architecture consists of three phases. The first phase includes data collection from various vital sign monitoring devices and different health record databases using Apache Kafka, a distributed messaging system [2], Apache Sqoop, designed for transferring data between relational databases, and Apache Hadoop [3]. In the second phase we have used BigData technologies for storing data, and for filtering and extracting useful information from large data volumes. For this purpose, we use MapReduce (MR) [4], which is designed to perform parallel data processing on large clusters, and Apache Hadoop [5], which is an open source framework written in Java that implements the MR model. Logistic regression with MR is used on filtered data to build a prediction model for detecting the emergency condition at an early stage. In the third phase, real-time processing is performed to predict emergency condition using the afore-mentioned prediction model. For real-time processing, we have used Apache Kafka and distributed stream processing engine, Apache Storm [6]. Using Apache Kafka and Storm, data streaming in real time can be conducted reliably. Data velocities of thousands of messages every second can be handled using Storm and Kafka.

We have evaluated the prediction performance on different prediction and observation windows for early detection of an emergency condition in real time on Vital Signs dataset of Queensland University, which was recorded from 32 patients under anaesthesia condition at the Royal Adelaide Hospital. We have also tested our prediction model for offline analysis on diabetes dataset, which was recorded during a 10 years (1999–2008) period at US hospitals to predict the risk of readmission. In healthcare application, there is a need to predict the health-related issues of patient. Logistic regression is used when target variable is categorical. In our work, logistic regression is used to predict the health-related issue by monitoring the vital signs of patient.

This paper is organised as follows. Section 2 presents a brief introduction of related literature, which includes the Hadoop system architecture, introduction of Kafka and Storm, and motivation for this study. Section 3 presents the details of each phase of the proposed architecture. Section 4 presents the experimental results. Finally, section 5 presents the conclusion of this study.

2. Related works

In this section, a brief introduction of related literature is given. The current healthcare system now focuses on changing its facilities from the traditional treatment to the prevention, early detection, prediction and warning. To evaluate the benefits of early detection of emergency in healthcare systems, a number of surveys, interviews and reviews have been conducted. In 2001, Agarwal *et al* [7] introduced a classifier based on Bayesian network to predict when a patient might have an emergency. In this work, they used machine learning and genetic algorithm to maximise the accuracy of classification results. In 2007, Ramon *et al* [8] described data mining algorithms decision trees, Naive Bayes, First Order Random Forests and Tree Augmented Naive to predict the emergency condition of patients in an intensive care unit (ICU). They also discussed the data mining challenges in healthcare for a large amount of data.

The rate of data generation in health informatics is very large. To store healthcare records of a large number of patients, storage of terabytes to petabytes is required. In the traditional approach, We have to work hard to store it, access it, manage it and process it. New forms of integration are required to uncover hidden values from large datasets that are complex, diverse and of a massive scale. Google has given a solution to this problem using an algorithm known as MR. In 2004, Dean *et al* [4] at Google Labs introduced a MR algorithm for simplified data processing on large clusters. MR technique runs on the large numbers of a cluster for parallel data processing and it is highly scalable. A typical MR technique uses thousands of machines and processes many terabytes of data at a time. The MR algorithm runs on two important tasks, as the name suggests, Map and Reduce. Map converts a set of data to another set of data where each element is further broken into key-value pairs called tuples. Reduce task takes the output of tuples from a map as input and combines into a smaller set of tuples. These tasks are always performed in a sequence, that is, first map and then reduce. At runtime, input data are divided into multiple data blocks of the same size. These same-sized data blocks are then assigned to nodes that perform parallel map function. After map function, the generated output is a datum composed of several $<key, value>$ pairs. This datum is then passed to nodes that perform reduce function and generates final output data. Figure 1 shows the MR flow chart.

Doug Cutting *et al* developed Hadoop [9], which is an Open Source project based on the solution provided by Google. In Hadoop, MR is used to run applications in which parallel data processing is required on huge amounts of data. Hadoop Distributed File System (HDFS) and MR are the two main components of Hadoop. On top of the

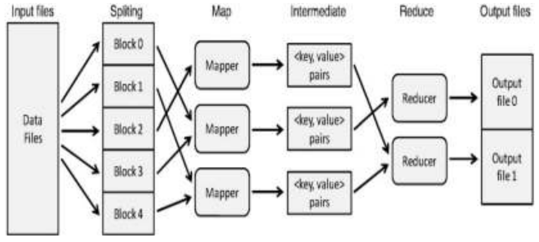


Figure 1. Overview of the MapReduce model.

local file system, HDFS supervises the data processing. Hadoop adopts master/slave architecture, in which a master node manages other slave nodes in the cluster. In the MR model, the master is called JobTracker, and each slave is called TaskTracker. The JobTracker is a java process, which is responsible for monitoring the job. JobTracker manages the Map/Reduce phase, and retries in case of errors. In the HDFS, the master is called NameNode, and each slave is called DataNode. Actual data are stored on DataNode. The file is split into a number of small blocks of equal size and then these blocks are store stored on different DataNodes. NameNode maintains the metadata of files. It stores information about a filesystem. Hadoop is a highly scalable storage platform. We can store very huge volumes of data across hundreds of data nodes and can access in parallel. Figure 2 shows an overview of the Hadoop framework.

In 2001, Vignesh and Sivasankar [10] proposed a framework that is based on Apache Hadoop to improve the healthcare informatics systems. They used a distributed Hadoop platform. It is deployable in different geographic locations at various healthcare centres. MR and Hadoop can be input/output intensive. It is not suitable for those applications that require a quick response on real-time data. The major challenges in real-time analysis are that real-time system should be always available to process all the real-time events; real-time system should be able to process millions of message per second, so it needs to be highly scalable and real-time system should be distributed in nature to process a large amount of data in a parallel manner. Kreps *et al* [11] developed a Kafka. It is a distributed messaging system for log processing. Kafka is distributed and scalable and offers high throughput; it allows applications to consume log events and also provides an API similar to the messaging system: the message producer, the message consumer and the message broker. It has categories called topics, which contain feeds of messages. Producers publish data to the topics of their choice. The producer chooses which message to assign to which partition within the topic. This can be done in a round-robin fashion to balance load or it can be done according to some semantic partition function, and processes the feed of published messages. Kafka performs as a cluster comprising one or more servers, each of which is called a broker. There can be multiple producers and consumers generating

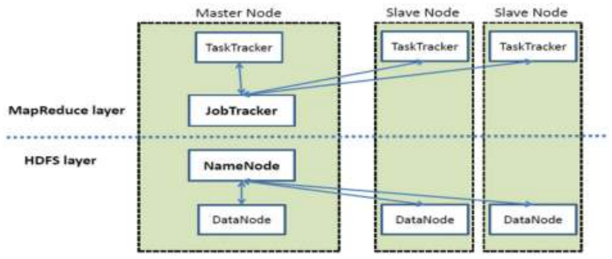


Figure 2. Overview of Hadoop architecture.

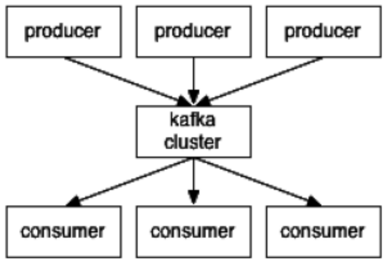


Figure 3. Overview of Apache Kafka.

messages and subscribing to a topic. A copy of each message for that topic is sent to each subscription. Figure 3 shows an overview of Apache Kafka. This design makes Kafka highly scalable and capable of processing millions of messages per second. Parallel processing can be done as a producer can write messages into the Kafka cluster and a consumer can consume messages from the cluster.

Twitter has developed a distributed real-time parallel processing platform called Apache Storm [6]. The Storm is an open source distributed system, which is used for real-time computation. Storm topologies are the combination of bolts and spouts. The topology where the data stream is injected is called spouts. The data streams that are piped into it are processed by bolts. Bolts can feed data from spouts or other bolts. Parallel processing of spouts, bolts and other data around is taken care of by Storm.

3. Healthcare BigData Analytical Architecture

The proposed architecture consists of three main phases. In the first phase, offline and real-time data are collected from various EHRs and monitoring devices. In the second phase, collected data are stored on HDFS; filtration is performed on this massive amount of data to extract potentially useful information and build prediction model with the help of filtered data using logistic regression. The third phase is for real-time processing to process real-time patients data collected from monitoring devices and perform actual prediction. Figure 4 shows an overview of our proposed architecture. In this section, we have presented the details of each phase of the proposed architecture.

3.1 *Collection of data*

In the modern healthcare system, a wide range of data is available, which can be helpful in predicting patient's risk for disease and in providing effective care. In our work, we have categorised patient data into two categories: real-time data and offline data. Real-time data include data obtained by continuously monitoring patient's vital signs, including body temperature, blood pressure, pulse rate, etc., from various monitoring devices. Offline data include demographic information of every patient, which includes information of birth date, birth place, gender, etc., historical

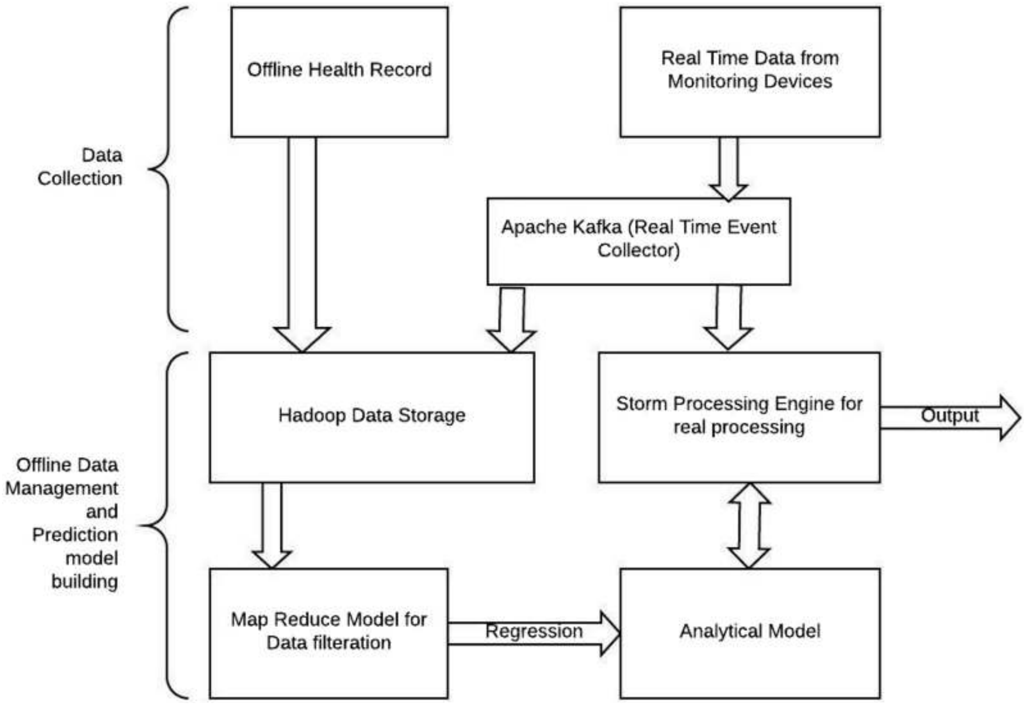

Data Collection
Offline Health Record
Real Time Data from Monitoring Devices
Apache Kafka (Real Time Event Collector)
Offline Data Management and Prediction model building
Hadoop Data Storage
Storm Processing Engine for real processing
Output
Map Reduce Model for Data filteration
Regression
Analytical Model

Figure 4. Real-time BigData and Predictive Analytical Architecture for healthcare application.

records of previous illnesses and treatments of every patient, which may be very useful in the future.

Apart from the data obtained from the patient, several other sources are also available, which are valuable for data analysis and decision making. In our work, Apache Sqoop is used to import offline data from various Public EHR databases into Hadoop and Apache Kafka is used to collect real-time data from the various monitoring devices at the Kafka cluster. In modern healthcare scenario, various monitoring devices are available, which can monitor vital signs in real time from anywhere using wireless or Bluetooth. As discussed in section 2, message producer, message consumer and message broker are the three major components of Kafka and producers publish data to the topics of their choice.

In our work, we have used devices monitoring vital signs as a Kafka producer. A separate Kafka producer is used for each patient and data of an individual patient are published to Kafka cluster with different topic names. For example, the topic name 'case1' is assigned to the first patient, 'case2' to the next patient and so on as shown in figure 5. The Kafka cluster retains all published data for a configurable period. After a datum is published, it is available for consumption; we use Hadoop and Storm as Kafka consumers in later phases of our architecture.

3.2 *Offline data management and prediction model building*

In this phase, offline and real-time data collected in the earlier phase are stored into HDFS in a parallel manner using the MR algorithm. In case of offline data, Apache Sqoop is used to import data from databases to HDFS as shown in figure 6.

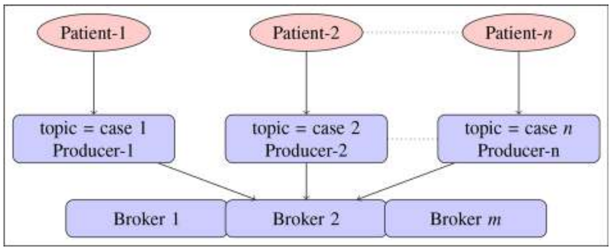

Patient-1
Patient-2
Patient-n
topic = case 1 Producer-1
topic = case 2 Producer-2
topic = case n Producer-n
Broker 1
Broker 2
Broker m

Figure 5. Collection of real-time patient data on Kafka.

Import using Sqoop is done in four steps:

1. Pull metadata from the database.
2. Submit MapReduce job.
3. Pull data from the database.
4. Write data to HDFS.

The imported data are stored in a directory on HDFS for further use. In the case of real-time data, Hadoop is used as a Kafka consumer. Hadoop-based consumer performs many map tasks to pull data from the Kafka cluster in parallel. In Hadoop, we can connect external data sources to Hadoop for parallel processing using MR. Kafka also provides MR layer that uses Record Reader and to read the records from the Kafka cluster, records are split into small blocks for parallel processing. In our case study, we have used open

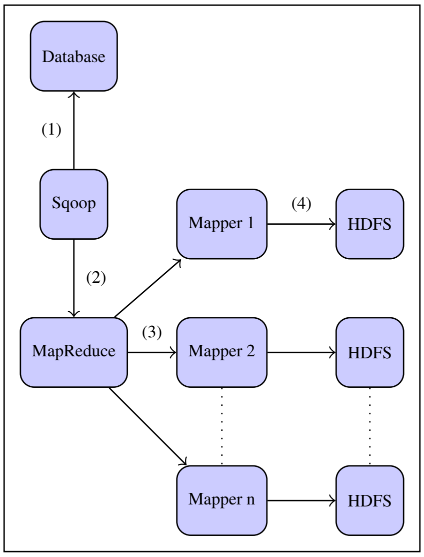


Figure 6. Import using Sqoop.

source Hadoop-Kafka consumer; it is available in GitHub [12]. For a given topic, Hadoop consumes data from Kafka. The Hadoop Kafka consumer keeps track message, which is consumed and maintained in HDFS. Following three steps are used to pull data from Kafka server into HDFS:

1. Produce test events to the Kafka server and generate offset files for each topic.
2. Submit MR job and assign number of mappers equal to the number of topic offsets.
3. Fetch data into HDFS from the Kafka server up to largest available offsets.

After data are stored into Hadoop, MR jobs are performed to filter data for extracting useful information from the large volumes of data. Later, these filtered data are used to build a prediction model. To build a prediction model, logistic regression is used with MR algorithm. Logistic regression is an algorithm to model the probability of an event as a function of another variable. Logistic function $x(z)$ that allows a single variable z can be expressed as

$$x(z) = \frac{1}{(1+e^{-z})}. \tag{1}$$

Let D be a dataset with binary outcomes. Every data point i contains a set of n attributes $z_{1,i}, z_{2,i}, \ldots, z_{n,i}$ (also called independent variables). The value of z_i is either O_i = 1 or O_i = 0. When outcome O_i = 1, then it is said to belong to the positive class, but when O_i = 0 then it belongs to the negative class. A regression model is created that classifies experiment z_i as positive or negative, that is, whether it belongs to positive class or negative class. The main aim of the logistic regression model is to provide a relationship between the attributes and the outcome so that the result of the logistic regression can be used to predict a new set of variables.

To find the relationship between each experiment z_i and its expected result, logistic function is used. This logistic function is based on the afore-mentioned regression function; it allows many variables and can be written as follows:

$$P(Z_i) = \frac{1}{\left(1+e^{-(\beta_0+\beta_1 z_{1,i}+\beta_2 z_{2,i}+\cdots+\beta_n z_{n,i})}\right)} \tag{2}$$

where $\beta_0, \ldots, \beta_n$ are regression coefficients. The regression coefficients $\beta_0, \beta_1, \ldots, \beta_n$ can be grouped into a single vector β of size $n+1$. Now, the aim is to find out the values of regression coefficient β such that classification with high accuracy can be obtained. An extra pseudo-variable $z_{0,i}$ is added, for every data point i, with a constant value 1, which corresponds to coefficient β_0. The resulting attributes $z_{0,i}, z_{1,i}, z_{2,i}, \ldots, z_{n,i}$ are then grouped into a single vector Z_i of size $n+1$. Linear predictor function can be written as follows:

$$f(\beta, Z_i) = \beta Z_i. \tag{3}$$

Thus, we can write the afore-mentioned equation as

$$P(z) = \frac{1}{\left(1+e^{-f(\beta,Z_i)}\right)} \tag{4}$$

and

$$f(\beta, Z_i) = \ln\left(\frac{P(Z_i)}{(1-P(Z_i))}\right). \tag{5}$$

Thus, our regression model is

$$O = P(Z_i) + e \tag{6}$$

where e is error term. We can think of an experiment in D as a Bernoulli trial with mean parameter $\mu(z_i)$. For a Bernoulli random variable O_i, mean is $\mu(z_i)$ and variance of the random variable will be $\mu(z_i)(1-\mu(z_i))$. There will be only two values for error e. If $O=1$ then $e = 1-\mu(z)$, otherwise $e=\mu(z)$.

The probability of the i^{th} experiment and the result in the database D, O, can be calculated as

$$P_{(Z_i,O_i|\beta)} = \begin{cases} P(Z_i) & \text{if } O=1 \\ 1-P(Z_i) & \text{if } O=0 \end{cases}$$

Therefore

$$P_{(Z_i,O_i|\beta)} = P(Z_i)^{O_i}(1-P(Z_i))^{1-O_i}. \tag{7}$$

From this expression we may derive likelihood and log-likelihood of the data D and outcome O with parameters β as

$$\mathbb{L}(Z, O, \beta) = \prod_{i=1}^{R} P(Z_i)^{O_i}(1-P(Z_i))^{1-O_i}, \tag{8}$$

$$\begin{aligned} &\ln \mathbb{L}(Z, O, \beta) \\ &= \sum_{i=1}^{R} \Big(O_i \ln P(Z_i) + (1 - O_i) \ln(1 - P(Z_i)) \Big) \\ &= \sum_{i=1}^{R} \ln(1 - P(Z_i)) + \sum_{i=1}^{R} O_i ln\left(\frac{P(Z_i)}{1 - P(Z_i)} \right) \\ &= \sum_{i=1}^{R} ln(1 - P(Z_i)) + \sum_{i=1}^{R} O_i f(\beta, Z_i) \end{aligned} \tag{9}$$

For estimation of maximum likelihood, log-likelihood function is differentiated with reference to the parameters and the value of derivative is set equal to zero.

$$\begin{aligned} &\frac{\mathrm{d}}{\mathrm{d}\beta_j}(\ln \mathbb{L}(Z, O, \beta)) \\ &= \sum_{i=1}^{R} \left(O_i \frac{\frac{\mathrm{d}}{\mathrm{d}\beta_j} P(Z_i)}{P(Z_i)} - (1 - O_i) \frac{\frac{\mathrm{d}}{\mathrm{d}\beta_j} P(Z_i)}{(1 - P(Z_i))} \right) \\ &= \sum_{i=1}^{R} \Bigg(O_i \frac{z_{j,i} P(Z_i)(1 - P(Z_i))}{P(Z_i)} \\ &\quad -(1 - O_i) \frac{z_{j,i} P(Z_i)(1 - P(Z_i))}{(1 - P(Z_i))} \Bigg) \end{aligned} \tag{10}$$

$$\frac{\mathrm{d}}{\mathrm{d}\beta_j}(\ln \mathbb{L}(D, O, \beta)) = \sum_{i=1}^{R} \big(z_{j,i}(O_i - P(Z_i)) \big) \tag{11}$$

where $j = 1, 2, \ldots, n$ and n is the number of attributes.

We are not able to set this to zero and solve exactly. Both the functions log-likelihood and likelihood are nonlinear functions in β and it cannot be solved analytically. Hence, numerical methods (such as Iteratively Re-weighted Least-Square (IRLS) technique) are typically used to find the maximum likelihood estimation. If we extend the definition such that $P(Z) = P(Z_1), P(Z_2), \ldots, P(Z_R)^T$, these equations can be written as

$$Z^T (O - P(Z) = 0. \tag{12}$$

Define

$$w_i = P(z_i)(1 - P(z_i)) \tag{13}$$

and $W = diag(w_1, \ldots, w_R)$. Each step of IRLS involves solving a weighted least squares. Step $i + 1$ involves

$$\begin{aligned} \hat{\beta_{i+1}} &= (Z^T W Z)^{-1} Z^T (W Z \hat{\beta_i} + (O - P(Z))) \\ &= (Z^T W Z)^{-1} Z^T W v \end{aligned} \tag{14}$$

where

$$v = (Z\hat{\beta_i} + W^{-1}(O - P(Z)). \tag{15}$$

To accomplish logistic regression, we have used MR job, where Map and Reduce functions collectively perform a weighted least-square regression based on the current coefficient values. The mapper computes a weighted sum of squares and cross-product for each chunk of input data shown in algorithm 1, and the reducer computes the regression coefficient estimates from the sums of squares and cross-products using algorithm 2. Once the prediction model is built we used it in both real-time and offline prediction.

Algorithm 1: Map function

```
1  Get all attributes into double array ;
2  Set int O ← class (either 0 or 1);
3  Assign a double array to vector $Z$;
4  if This is the first iteration then
5      Compute starting values $P(z_i)$ from Equation-6 ;
6      Derive $f(\beta, Z)$ values using Equation-5;
7  else
8      Compute $f(\beta, Z)$ as the linear combination using
         the current coefficient using Equation-3;
9      derive mean probabilities $P(Z_i)$ from equation-4;
10 end
11 Compute weight $w_i = P(Z_i) * (1 - P(Z_i))$
     (Equation-13);
12 Compute $v = f(\beta, Z) + \frac{O_i - P(Z_i)}{w_i}$ (Equation-15);
13 Compute matrix of unweighted data X = [v,Z];
14 Calculate weighted cross-products
     $wcp = X1' * W1 * X1$;
15 Assign output key-value as $< key, wcp >$;
```

Algorithm 2: Reduce function

```
1  Set variable owcp = 0;
2  while key has next value do
3      variable nwcp = value ;
4      owcp= owcp + nwcp;
5  end
6  Compute coefficients estimates from owcp;
7  Return the vector of coefficient estimates;
```

4. Real-time processing and actual prediction

In this phase, real-time events can get back by Storm processing engine from Kafka and perform actual prediction using the prediction model build in an earlier phase. In Storm, a stream of tuples are processed in parallel and for this, we are required to introduce our spouts and bolts, and also Storm is configured so that it can be optimised for its parallel processing. If Storm is required to collect the data from Kafka cluster and data are processed in parallel, a Storm-Kafka spout is required. Messages are read from the Kafka with the help of a spout; this data stream is sent to Storm bolts for the downstream processing. Storm-Kafka spout is configured, and data can now be read from the specified Kafka topic. Now Storm will start accessing the message. To process the tuple streams that can be accessed from Kafka, another bolt is needed to be configured. In our work, we have used a topology of Storm-Kafka spout for

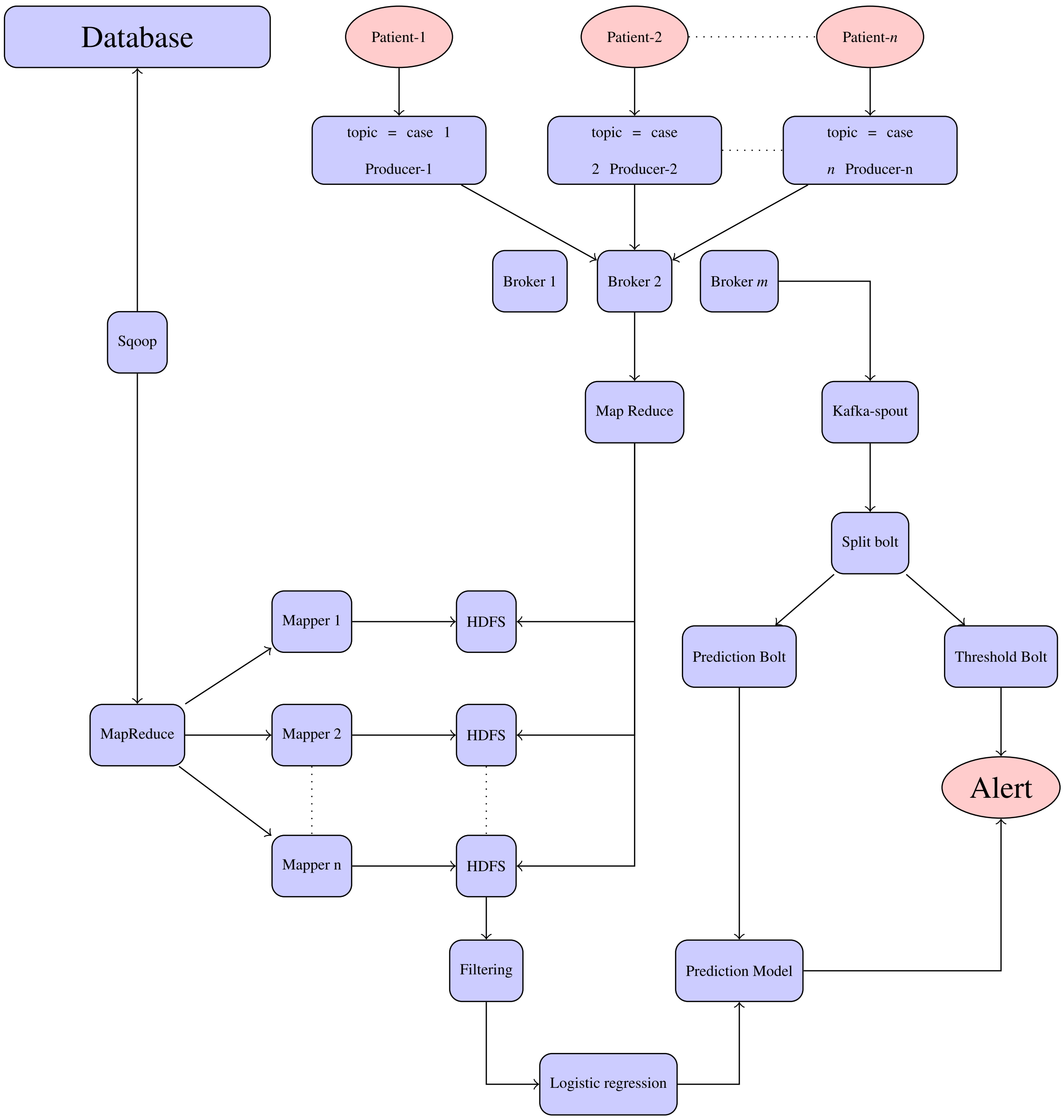


Figure 7. Work flowchart.

each topic, several bolts to check threshold value, to generate alert and to predict the emergency condition using the prediction model. In the real-time processing part, we have designed a java-based simulator that publishes data of each patient with different topic names into Kafka after every 10 ms. To read data from Kafka we have created a Storm topology of one Storm-Kafka spout, one threshold bolt and one prediction bolt for each topic. Kafka-spout reads data from Kafka and emits this to the threshold bolt and prediction bolt. For example, we have assigned topic name case 1 to patient 1, so Storm Kafka spout of topology 1 fetches data from Kafka server of topic name case 1 and emits this to the threshold bolt and prediction bolt of topology 1. Threshold bolt split received the message into separate fields such as blood pressure, pulse rate, respiratory rate, etc. and compares the value of each field to a predefined threshold range of respective field; if this value is out of range, then an alert is generated. For example, the threshold range of heart rate is 50–100. If the value in heart rate field is not in a given threshold range, then an alert will be generated. Prediction bolt classifies the message collected from spout using the prediction model, and if the result of the prediction model is positive then prediction bolt generates an alert. Real-time processing is shown in Algorithm 3. We have used Kafka and Storm to process real-time data and persist this data to further offline process in HDFS. Algorithm 3 describes using Kafka that producer is responsible for publishing records to Storm for real-time processing and persisting records into HDFS for offline processing. Producer publishes the 100 records into HDFS as mentioned in Algorithm 3 when the value of count is 100. The count value is taken as a parameter; if this value is large, it increases the publish time of producer. If count is much less, then persisting time will increase as the persisting process will be used frequently in very less time span. Hadoop cluster decreases the training time of big data using MR as algorithms run in each system of cluster in parallel. Flow chart of the work is shown in figure 7.

Algorithm 3: Real-time processing.

```
KAFKA:
Set Kafka Producer for each vital sign monitor;
for each monitoring device do
    Set count = 1;
    Set hadoop offset=1;
    while device is on do
        if count=100 then
            Write record into HDFS form Hadoop
              offset to read the last offset using Map
              Reduce algorithn;
            Set count = 1;
            Set Hadoop offset=last read offset;
        end
        Set field count with records;
        Publish record with a unique topic name at
          Kafka cluster ;
        Increment count by 1;
    end
end
STORM:
Configure Kafka spout by specifying the topic name,
  zookeeper hosts and offset;
for each topic do
    Set Start Offset to -1 (To read the new messages
      from the last read offset);
    Create a topology of 1 kafka-spout and 3 bolt
      (Split,threshold,prediction);
    Collect record from kafka using kafka-spout and
      emit it to split bolt;
    Split record in to field by split bolt;
    if count filed is equal to 100 then
        Emit data to prediction bolt;
        Calculate probability of emergency condition
          using logistic regression prediction model;
        if probability of emergency is greater than
          0.8 then
            Alert;
        end
    else
        Emit data to threshold bolt;
        Compare value of each field from predefined
          threshold range;
        if value is not in threshold range then
            Alert;
        end
    end
end
```

5. Experiment and result

In this section, we have presented experiments performed using the proposed architecture for real-time as well as offline analysis to perform several prediction tasks on different datasets and results.

5.1 *Experimental environment*

The experimental environment is shown in table 1. In this three systems are used; each one has 4 GB RAM, and 500 GB Storage disk. To store data and to implement MR algorithm we use Hadoop version 2.7.0. For real-time processing we use Kafka-0.8.2.1 and Storm-0.11.0. To build prediction model using regression we use Mahout-0.11.0. We create one namenode and three datanodes. All the machines use Ubuntu 15.04 as their operating system.

5.2 *Real-time analysis*

In the real-time analysis, Vital Signs dataset taken from the University of Queensland was used, which was recorded from 32 patients for early detection of emergency conditions such as bradycardia, tachycardia, hypertension, hypotension and hypoxemia using Vital signs such as heart rate, blood pressure, pulse rate, respiratory rate and oxygen saturation (SpO2). Out of 32 cases of the dataset, we have used 20 cases to build a prediction model. Table 2 shows the information of data attributes used in this case.

Table 1. Experimental environment.

	Specification	Amount
Machine	4-GB memory; 500-GB disk	1 master; 3 slaves
Hadoop version	Hadoop 2.7.0 (stable version)	
Kafka version	Kafka-0.8.2.1	
Storm version	Storm-0.11.0 (stable version	
Sqoop version	Sqoop-1.4.6 (stable version)	

Table 2. Information about data attributes.

Parameter	Normal range for adults
Heart rate	60–100 (beats per minute)
Blood pressure (systolic)	90–120 (mmHg)
Blood pressure (diastolic)	60–80 (mmHg)
Pulse rate	60–100 (beats per minute)
Oxygen saturation (SpO2)	94–99 (%)
Respiration rate	12–16 (breaths per minute)

Every vital sign has its importance in detecting an emergency. For example, in the case of hypotension and hypertension, blood pressure has more importance than the rest of the vital signs. Similarly, the heart rate has more importance when considering bradycardia and tachycardia. The relationship between vital signs and different disease is shown in table 3. For detection of a single disease, we have considered all the vital signs using a logistic regression model that gives a high level of accuracy and reliable results for any health event. For example, hypotension can be predicted only by 'low blood pressure' without considering other parameters. In our work, we have considered all the vital signs instead of taking only blood pressure for classification to get more accurate results.

We built models for bradycardia, tachycardia, hypertension, hypotension and hypoxemia prediction. To build prediction model, we used logistic regression with MR on filtered data. To filter data, two MR jobs are used. In the first job, we extract vital signs of each patient after 1 min; in the second job, we combine the extracted vital signs up to size of observation window. To train our models, we have used extracted data as attributes.

Table 4 shows the total number of alarms generated by the proposed system on different predictions and an observation window for early detection of bradycardia, tachycardia, hypertension, hypotension and hypoxemia. In table 5, for bradycardia and hypertension physical sign, accuracy increases as the size of the observation window increases. In the case of physical sign tachycardia, hypotension and hypoxaemia, accuracy increases when the prediction window increases from 2 to 5, and it also increases when the prediction window size is fixed at 10 and observation window size increases from 15 to 20. The highest accuracy is achieved when the observation window size is 20 in all the 5 physical signs mentioned in table 5. The maximum accuracy achieved is 80.8%, 82.8%, 81.8%, 82.4% and 79.6% for the physical sign of bradycardia, tachycardia, hypotension, hypertension and hypoxaemia, respectively. We have compared our online prediction model with Artificial Neural Network and Support Vector Machine as mentioned in [13] and [14]. The comparison results are shown in table 5. As MR is used with logistic regression in our model, it helps in reducing the training time for the model as mentioned in figure 9.

Table 3. Relationship between vital signs and different diseases.

Physical sign	Heart rate	BP (systolic)	BP (diastolic)	SpO2	Pulse rate
Bradycardia	Below 60	–	–	–	–
Tachycardia	Above 100	–	–	90–99%	–
Hypotension	–	Below 90	Below 60	–	50–100
Hypertension	–	Above 120	Above 80	–	60–100
Hypoxaemia	–	–	–	Below 90%	–

Table 4. Total number of alarms generated by the proposed system on different prediction and observation windows.

Physical sign	Observation window	Prediction window	No. of alerts	Accuracy	Precision	Recall	AUC	*F*-score
Bradycardia	5	2	28	0.732	0.687	0.557	0.921	0.631
	10	5	19	0.752	0.723	0.621	0.942	0.676
	15	10	11	0.778	0.735	0.626	0.952	0.673
	20	10	12	0.808	0.757	0.692	0.972	0.683
Tachycardia	5	2	32	0.762	0.707	0.587	0.941	0.641
	10	5	23	0.802	0.773	0.671	0.952	0.696
	15	10	14	0.798	0.755	0.656	0.955	0.693
	20	10	16	0.828	0.737	0.672	0.942	0.677
Hypotension	5	2	24	0.722	0.677	0.547	0.911	0.621
	10	5	15	0.782	0.733	0.631	0.922	0.656
	15	10	7	0.758	0.715	0.606	0.932	0.653
	20	10	8	0.818	0.777	0.702	0.962	0.695
Hypertension	5	2	38	0.762	0.707	0.577	0.951	0.651
	10	5	29	0.772	0.733	0.631	0.932	0.656
	15	10	20	0.798	0.755	0.646	0.966	0.684
	20	10	22	0.824	0.761	0.660	0.955	0.689
Hypoxaemia	5	2	26	0.712	0.675	0.535	0.901	0.609
	10	5	17	0.772	0.723	0.631	0.922	0.656
	15	10	9	0.758	0.713	0.604	0.930	0.651
	20	10	11	0.796	0.735	0.670	0.951	0.675

Table 5. Accuracy (%) of experimental results on real-time datasets.

Datasets	Logistic regression with MR	ANN	SVM
Bradycardia	80.8	80.2	79
Tachycardia	82.8	80.0	80
Hypotension	81.8	81.8	78
Hypertension	82.4	82.0	81
Hypoxaemia	79.6	79.6	84

$$Accuracy = \frac{no.\ of\ correct\ predictions}{total\ no.\ of\ test\ cases} \tag{16}$$

$$Precision\,(p) = \frac{TP}{TP + FP} \tag{17}$$

$$Recall\,(r) = \frac{TP}{TP + FN} \tag{18}$$

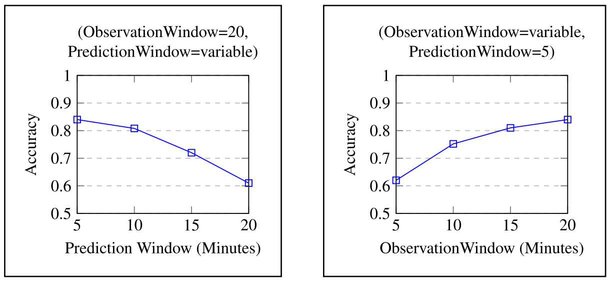
$$F - score = \frac{2pr}{p + r} \tag{19}$$

$$Sensitivity = \frac{TP}{TP + FN} \tag{20}$$

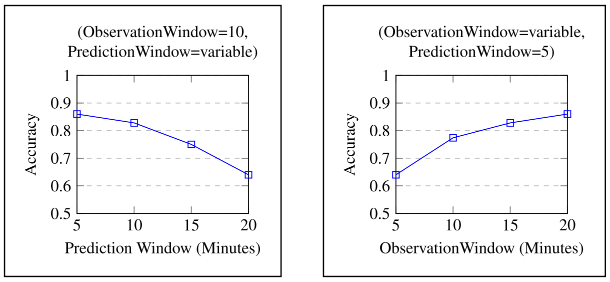
$$Specificity = \frac{TN}{TN + FP} \tag{21}$$

where *TN* is true negative, *TP* is true positive, *FP* is false positive and *FN* is false negative.

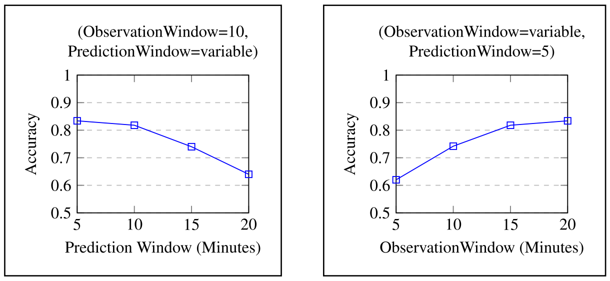
Figure 8 shows accuracy for bradycardia, tachycardia, hypertension, hypotension and hypoxemia prediction on different prediction and an observation windows. We can observe from figure 8 that as we increase prediction window while keeping observation window constant, the accuracy of disease decreases and if we increase observation window while keeping prediction window constant, accuracy also increases.



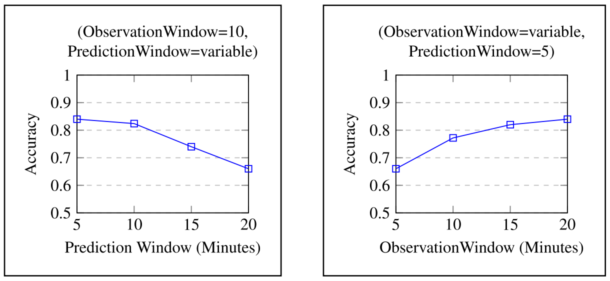
(**a**) Bradycardia prediction



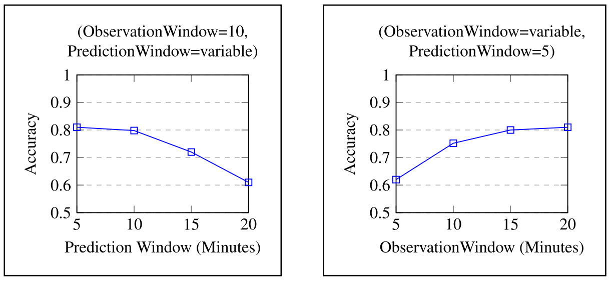
(**b**) Tachycardia prediction



(**c**) Hypotension prediction



(**d**) Hypertension prediction



(**e**) Hypoxaemia prediction

Figure 8. Accuracy for bradycardia, tachycardia, hypertension, hypotension and hypoxemia prediction on different prediction and observation windows.

Table 6. Information of prediction test on different datasets.

Dataset information	No. of instances	No. of attributes	Task
Diabetes	101745	43	Predicts risk for readmission
Breast cancer	536	32	Predicts risk for breast cancer
Heart disease	281	14	Predicts risk for heart disease

5.3 *Offline analysis*

For offline analysis, we have performed several prediction tasks on different datasets, which are shown in table 6. Figure 10 shows the comparison of accuracy using regression with MR and without MR and figure 9 shows the comparison in of performance time using regression with MR and without MR. Accuracies for offline datasets Heart disease, Breast cancer and Diabetes have been shown in figure 10. The best accuracy found in our experimentation for Heart disease, Breast cancer and Diabetes prediction is 96.8%, 94.2% and 75%, respectively, with MR. From the graph shown it can be observed that using MR algorithm there is an increase in accuracy for large data but not considerable change in accuracy for small dataset. However, execution time becomes almost half with MR.

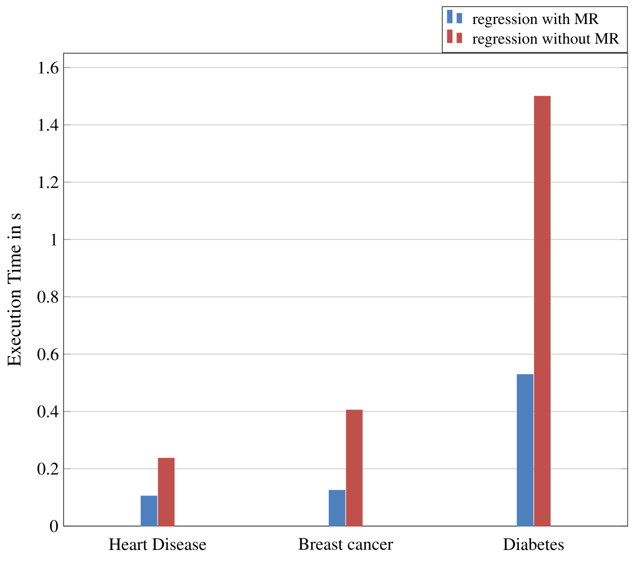


Figure 9. Execution time.

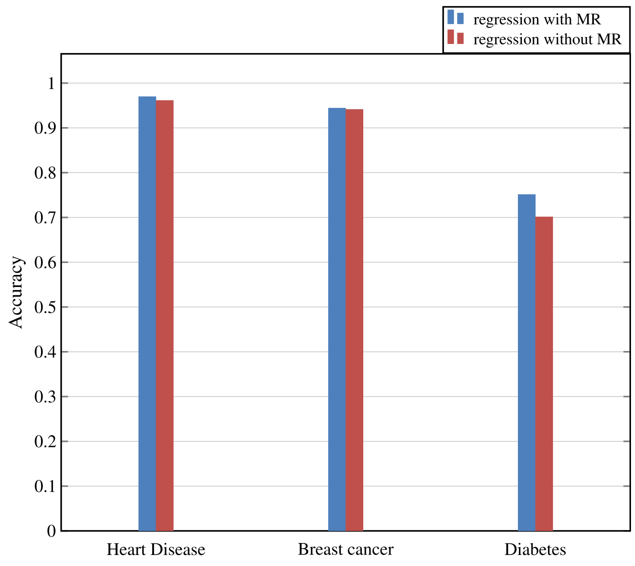


Figure 10. Average accuracy of expected values after 10 min.

6. Conclusion

Real-time analysis is required for continuous monitoring of vital signs to examine any change in patient's condition. Monitoring devices generate a huge amount of data and processing of these data in real time is a difficult task. In this paper, we have proposed a real-time BigData and Predictive Analytical Architecture for healthcare applications. The proposed architecture can perform early detection of emergency in real time and can analyse structured and unstructured data to predict patient's risk for disease or readmission. We have used Apache Hadoop and logistic regression with MR for risk prediction. To monitor patient's vital signs and predict an emergency, Kafka and Storm are used, which provide real-time streaming for fast moving data as Storm and Kafka can handle data velocities of tens of thousands of messages every second. We have evaluated prediction performance on different benchmark datasets for detection of an emergency condition in real time and possibility of readmission of any patient, and this performance is evaluated on different prediction and observation windows. Experimental results demonstrate the effectiveness of our proposed architecture.

References

[1] Abaker I, Yaqoob I, Khan S and Mokhtar S 2015 The rise of BigData on cloud computing: review and open research issues. *Information Systems* 47(3): 95–115
[2] KApache-Kafka http://kafka.apache.org/
[3] Apache Sqoop http://sqoop.apache.org/
[4] Dean J and Ghemawat S 2004 MapReduce: simplified data processing on large clusters. In: *Proceedings of OSDI*, pp. 137–150

[5] Apache http://httpd.apache.org/
[6] Storm http://storm-project.net/
[7] Agarwal N, Liu H and Zhang J 2001 Using Bayesian networks in the construction of a bi-level multi-classifier: a case study using intensive care unit patients data. *Artificial Intelligence in Medicine* 22: 233–248
[8] Ramon J, Fierens D, Fabia G, Meyfroid G, Blockeel H and Bruynooghe M 2007 Mining data from intensive care patients. *Advanced Engineering Informatics* 21: 243–256
[9] Hadoop http://hadoop.apache.org/
[10] Vignesh R and Sivasankar E 2001 Modern framework for distributed healthcare data analytics based on Hadoop. *International Federation for Information Processing* 22: 233–248
[11] Kreps J, Narkhede N and Rao J 2011 Kafka: a distributed messaging system for log processing. In: *Proceedings of the 6th International Workshop on Networking Meets Databases (NetDB)*, Athens, Greece
[12] Hadoo-Kafka consumer *https://github.com/kafka-dev/kafka/tree/master/contrib/hadoop-consumer*
[13] Burke H B, Goodman P H, Rosen D B, Henson D E, Weinstein J N, Harrell F E, Marks J R, Winchester D P and Bostwick D G 1997 Artificial neural networks improve the accuracy of cancer survival prediction. *Cancer* 4: 857–862
[14] Yu W, Liu T, Valdez R, Gwinn M and Khoury M J 2010 Application of support vector machine modeling for prediction of common diseases: the case of diabetes and pre-diabetes. *BMC Medical Informatics and Decision Making*