Accepted Manuscript

SLA based healthcare big data analysis and computing in cloud network

Prasan Kumar Sahoo, Suvendu Kumar Mohapatra, Shih-Lin Wu

PII: S0743-7315(18)30245-4
DOI: https://doi.org/10.1016/j.jpdc.2018.04.006
Reference: YJPDC 3869

To appear in: *J. Parallel Distrib. Comput.*

Received date : 1 December 2017
Revised date : 4 March 2018
Accepted date : 6 April 2018

Please cite this article as: P.K. Sahoo, S.K. Mohapatra, S.-L. Wu, SLA based healthcare big data analysis and computing in cloud network, *J. Parallel Distrib. Comput.* (2018), https://doi.org/10.1016/j.jpdc.2018.04.006

This is a PDF file of an unedited manuscript that has been accepted for publication. As a service to our customers we are providing this early version of the manuscript. The manuscript will undergo copyediting, typesetting, and review of the resulting proof before it is published in its final form. Please note that during the production process errors may be discovered which could affect the content, and all legal disclaimers that apply to the journal pertain.

*Highlights (for review)

ACCEPTED MANUSCRIPT

1. SLA based healthcare big data analytic architecture is designed to process both batch and streaming patient data in Spark platform.
2. Ranking of patient's data based on SLA, patients health condition, disease severity and emergency situation is made to improve the processing speed.
3. Efficient data distribution mechanism is designed to allocate both batch and streaming data among the Spark worker nodes.
4. Priority based job allocation algorithm is designed to allocate jobs with minimum inter-network latency and processing time.
5. Probabilistic Semi-Naive Bayes algorithm is designed to analyze and predict the future health condition of the patients taking inter-dependency among healthcare parameters. Besides, a dimension reduction algorithm is designed to reduce the input healthcare parameters for improving accuracy of prediction.

*

ACCEPTED MANUSCRIPT

SLA Based Healthcare Big Data Analysis and Computing in Cloud Network

Prasan Kumar Sahoo[a], Suvendu Kumar Mohapatra[b], Shih-Lin Wu[a,*]

[a] *Dept. of Computer Science and Information Engineering, Chang Gung University, Kwei-Shan, 333, Taiwan (ROC)*
[b] *Industry 4.0 Implementation Center, National Taiwan University of Science and Technology, Keelung Rd., Da'an Dist., Taipei City 106, Taiwan (R.O.C.)*

Abstract

Large volume of multi-structured and low-latency patient data are generated in healthcare services, which is challenging task to process and analyze within the **Service Level Agreement (*SLA*)**. In this paper, a Parallel Semi-Naive Bayes (PSNB) based probabilistic method is used to process the healthcare big data in cloud for future health condition prediction. In order to improve the accuracy of *PSNB* method, a Modified Conjunctive Attribute (MCA) algorithm is proposed for reducing the dimension. Emergency condition of the patient is considered by setting a global priority among the patients and an Optimal Data Distribution (ODD) algorithm is proposed to position both batch and streaming patient data into the Spark nodes. Further, a Dynamic Job Scheduling (DJS) algorithm is designed to schedule the jobs efficiently to the most suitable nodes for processing the data taking *SLA* into account. Our proposed *PSNB* algorithm provides better accuracy of 87.8% for both batch and streaming data, which is 12.8% higher than the original Naive-Bayes (NB) algorithm and can conveniently be employed in various patient monitoring applications.

Keywords: Big Data, cloud computing, healthcare, spark.

*Corresponding author.
Email addresses: pksahoo@mail.cgu.edu.tw (Prasan Kumar Sahoo), suvendukm@mail.ntust.edu.tw (Suvendu Kumar Mohapatra), slwu@mail.cgu.edu.tw (Shih-Lin Wu)

ACCEPTED MANUSCRIPT

1. Introduction

Digital revolution such as Internet of Things (IoT) [1], Wireless Body Networks (WBNs) [2], Big Data [3] and Cloud Computing [4] enables the day-to-day living style easier and better. Big Data deals with extremely large data sets having four different characteristics including Volume, Variety, Velocity and Veracity. Besides, ceaseless streams of healthcare data are generated in large volume by ubiquitous smart devices such as smart phone, pulse oximeters and body sensors on real-time patient monitoring. Under the existing solution methods, it is very difficult to analyze and process both streaming and batch data together in a single platform within the deadline. As a result, the first problem is how to reduce the dimension of the health parameter for better accuracy. The second problem is how to find the dependencies among the healthcare parameters and priority of the patients based on the influential parameters. Thirdly, which appropriate method can be used for analysis and processing of those multi-structured, low-latency patient data with higher accuracy and efficiency. By considering above-mentioned issues, Big Data analysis, and processing are two major challenges in the sizable healthcare industry.

In data analysis, various classification [5], clustering [6] and predictive analytic [7] algorithms are used based on the input and output data sets. However, many of those tools are outdated [8] as they are unable to handle large volume of multi-structured healthcare data sets. Specifically, in healthcare, the patient data are not only large in volume but also are generated with a tremendous speed, which requires an advanced platform for both analysis and processing. Also, the health condition of a patient is always related with some uncertain factors based on the clinical parameters. For probabilistic approach, Naive Bayes (NB) [9] is the best and most popular algorithm due to its efficiency. However, *NB* algorithm can be applied only on the independent data sets, which is not suitable for healthcare big data as most of the data have dependency among multiple parameters. Hence, the Semi-Naive Bayes (SNB) [10] algorithm can be used, which allows certain degree of dependency on input parameters.

In healthcare applications, missing of any *SLA* [11] has highest impact on emergency patient data analysis due to the severity of the disease. The emergence of Big Data demands a distributed environment with parallel and fast computation. Hence, many big data processing platforms such as Hadoop [12] and Apache Storm [13] are mostly used for batch and streaming data

ACCEPTED MANUSCRIPT

processing, respectively. However, the Storm platform is found to be time-consuming with low throughput [14] for processing both batch and streaming data altogether. Healthcare data normally contain both streaming and batch data [15] for analysis and processing. Therefore, Apache Spark [16] can be used as our processing platform to process and analysis of both streaming and batch data altogether in a single API, which is time efficient. In Spark, Resilient Distributed Datasets (RDDs) [17] are used for efficient data sharing during parallel computation, which can enhance the overall system throughput.

Cloud computing is the most promising technology used in healthcare for distributed storage and processing of patient data with help of virtualization. Though, some analytic models such as BStream [18] are proposed for the bursty input and over-provisioning by using internal cloud to an external one, it is limited to the processing time, fault and straggler tolerance during execution [14] as Storm model is used for the processing. Therefore, it is highly essential to employ a suitable resource scaling and management scheme in Spark environment to satisfy the SLA. In [19], authors primarily focus on the Data Center network traffic prediction. However, the traffic prediction for an external cloud is not considered by the authors, which can affect both network utilization and congestion. Therefore, Inter-cloud Data center (ID) and External-cloud Data center (ED) are incorporated into a single environment to handle and process the colossal amount of both streaming and batch data within the SLA. The external cloud is adopted for hard deadline based jobs and load balancing in the internal cloud to satisfy the SLA.

Inefficient scheduling of jobs in the worker nodes may lead to failure of processing within SLA [20]. Hence, it is essential to schedule the jobs to the most suitable nodes for processing by minimizing the job completion time and satisfying the SLA in a multi-cloud environment. In this paper, we address all the above issues to mitigate the processing delay with low network latency and satisfy the SLA in a multi-cloud environment, which has significant improvement on patient data analysis and processing.

Rest of the paper is organized as follows. Related works on big data analytic and processing are discussed In Section 2. Problem formulation of our work is given in Section 3. Healthcare data processing mechanism is presented in Section 4. A probabilistic Big Data analytic mechanism is proposed in Section 5. Performance evaluation of our proposed models is given in Section 6 and concluding remarks are made in Section 7.

ACCEPTED MANUSCRIPT

2. Related Work

A comprehensive study has been carried out on healthcare big data analysis, processing and dimension reduction using various distributed parallel processing methods. In [21], authors discuss the recent developments in healthcare big data. In [10], Backward Sequential Elimination and Joining (BSEJ) method is proposed for applying the dependencies in classifying instances. However, the time and space complexity is very high for BSEJ algorithm. In [22], a dimension reduction method is proposed for the improved image registration of high-dimensional data, which combines both image pair and detailed texture. In [23], a dimension reduction mechanism is introduced by the authors, where pruning is done based on the information gain ratio of the decision tree to improve the accuracy of the proposed PRF algorithm. However, the tree building time is so high, which increases exponentially with increase in height of a tree. Hence, a Modified Conjunctive Attribute (MCA) algorithm is proposed in our work for dimension reduction of the healthcare data to improve the accuracy of Semi-Naive Bayes (SNB) algorithm.

In Big Data processing, Storm [13] is commonly used for near real-time streaming processing. Mostly, Storm platform is found costly in terms of processing time and communication delay due to different frameworks. Hence, the in-memory, cluster computing Spark [24] Streaming platform is used as our processing model for the near-real time streaming and batch healthcare data processing. In any parallel processing environment, data locality is one of the most important performance bottlenecks as missing of any partitions of the data block during execution leads to processing delay. In [25], a data-locality-aware scheduler is proposed for guaranteeing the data locality. In [26], splitting and combination algorithm for skew intermediate data blocks (SCID) method is proposed for data placements in Spark environment to improve load balancing for reduce tasks. However, SCID method takes more time as sampling and sorting are performed. Even, the intermediate results are fetched from a specific bucket, which lead to a bottleneck situation. Hence, an adaptive Optimal Data Distribution (ODD) algorithm is proposed in this paper to overcome the above data locality issues.

In [27], a sub-task scheduling framework "Millipedes" is proposed for Yet Another Resource Negotiator (YARN) including MapReduce and Spark, where each subtask is allocated to the nodes by the local scheduler depending on the resource usage. However, there is no consideration of overall job completion as the total jobs need to be finished within a certain deadline, i.e.

SLA. In [28], a job scheduling algorithm is proposed by the authors, where jobs are assigned to the nodes based on data locality using delay scheduling. However, scheduling of priority jobs with *SLA* is not considered in the existing schedulers. **In [29], a scheduling mechanism is proposed for MapReduce jobs. However, the impact of distort data set on the execution time of jobs is not considered by the scheduler. In [30], a distributed scheduling algorithm is designed to schedule the real-time skewed MapReduce jobs. However, a high performance overhead is incurred due to repartitioning and prediction of partition size.**

In [31], a detailed feature analysis of big data schedulers is addressed, where scheduler latency is found to be the most important performance characteristic of the scheduler. In [32], a balanced resource scheduling mechanism is proposed to minimize the resource cost in multi-cloud environment. However, the communication time is higher in a multi-cloud environment, which induces the processing delay for healthcare patient data. Almost, all existing models have higher processing time for the healthcare prioritized jobs in an emergency condition. Hence, a Dynamic Job Scheduling (DJS) algorithm is proposed to address the key underlying issues such as processing of batch and streaming data based on priority of the jobs.

Prediction models play a vital role for future disease prediction upon analyzing the large volume of healthcare data. In [33], authors propose the disease prediction model by using different types of artificial neural networks (ANNs). However, ANN has higher processing latency as random weights are associated with each layer during the training of the model. Any small change in the input data set has a visible impact on the model that results the unstable output. A predictive model is proposed in [34] by using the ECG features and Naive Bayes classifier for ventricular arrhythmia disease. However, the clinical data sets such as blood pressure, chest pain, etc. are not considered as the input parameters. Moreover, the dependency among the input parameters cannot be considered in the Naive Bayes classifier. Hence, a Parallel Semi-Naive Bayes (PSNB) based probabilistic method is planned in this paper for healthcare Big Data analysis.

2.1. Motivation and Contributions

Most of the data analytic and processing mechanisms are prone to delay when the input data volume and velocity are very high even though the inter-

ACCEPTED MANUSCRIPT

dependencies among the input healthcare parameters are not considered. Hence, it is highly essential to design a dynamic analytic algorithm for both batch and realtime healthcare Big Data. Basically, for processing real-time streaming data, Apache Storm is used. However, processing of both batch and streaming data is found to be time consuming with low throughput as two different APIs are used by Storm. In the medical application, some emergency patients data need to be processed in priority basis. To the best of our knowledge, no work considers the current health condition, disease severity, emergency factor and *SLA* level altogether to process the healthcare big data in a priority basis. Hence, priority of the patient's emergency data is considered along with the above constraints to process the data in Spark platform.

Furthermore, processing delay occurs during job scheduling. In Spark, FIFO and FAIR schedulers are available for concurrent queries. Basically, FIFO is the default scheduler in Spark for standalone mode and first job is executed with highest priority over all other jobs. However, the first execution of large processing time has significant delay impact on the subsequent job executions. To overcome the FIFO problem, FAIR scheduler is introduced in Spark 0.8 version, which is the best scheduler in multi-processing environment. All the jobs are executed in a Round Robin manner in FAIR scheduler, where all jobs get an equal chance for execution. However, the FAIR scheduler does not consider the resource constraints such as processing core, available memory, network bandwidth and CPU utilization of the workers, which lead to delay in total job completion time. Thus, the existing scheduling mechanisms cannot be applied to emergency patient streaming data analysis in healthcare environment. Hence, a *DJS* algorithm is proposed to schedule both prioritized batch and streaming data. The major contributions of our work can be summarized as follows.

- *SLA* based healthcare big data analytic architecture is designed to process both batch and streaming patient data in Spark platform.
- Ranking of patient's data based on *SLA*, patients health condition, disease severity and emergency situation is made to improve the processing speed.
- Efficient data distribution mechanism is designed to allocate both batch and streaming data among the Spark worker nodes.

- Priority based job allocation algorithm is designed to allocate jobs with minimum inter-network latency and processing time.
- Probabilistic Semi-Naive Bayes algorithm is designed to analyze and predict the future health condition of the patients taking inter-dependency among healthcare parameters.
- Algorithm is designed to reduce dimension of the input healthcare parameters for improving accuracy of prediction.

3. Problem Formulation

Let us consider a hybrid healthcare multicloud environment, where h number of hospitals are present in a set $H = \{H_1, H_2, ..., H_h\}$. Each hospital is coupled with different users such as doctors, outpatients and Body Area Networks (BAN) patients as shown in Fig. 1. Here, the outpatients are referred to as the patients who attend the hospital for treatment without staying there for treatment. Similarly, BAN patients are referred to as the chronic disease patients with smart sensors to monitor their health conditions round the clock. All the users act as the data sources of healthcare Big Data platform.

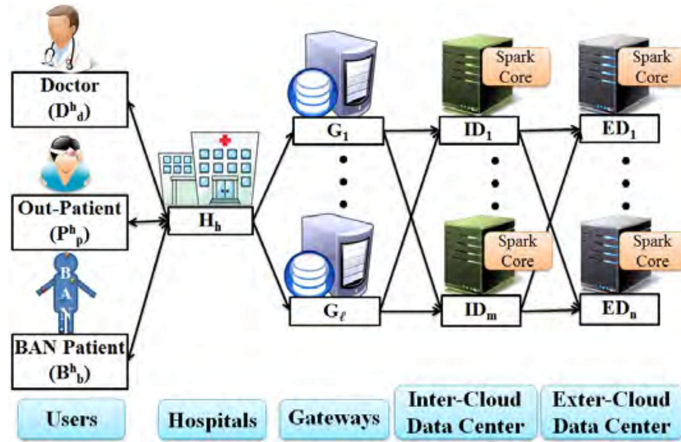

Figure 1: Proposed data source and processing model in multicloud.

ACCEPTED MANUSCRIPT

3.1. System Model

Let, d be the number of doctors present in a set D_i^h, where $i = \{1, 2, ..., d\}$ of hth hospital, $\forall h \in H$. Thus, $D_i^h = \{D_1^h, D_2^h, ..., D_d^h\}$, $\forall i \in D$. For example, D_2^3 represents the doctor 2 that belongs to the hospital 3. Let, p be the numbers of outpatients present in h^{th} hospital which can be represented in a set P_i^h where, $P_i^h = \{P_1^h, P_2^h, ..., P_p^h\}$, $\forall i \in P$ and $\forall h \in H$. For example, P_1^2 represents the patient 1 in hospital 2. In addition to the outpatients, BAN patients are also available with chronic disease and also registered in a hospital. Similarly, let b be the number of BAN patients present in a set B_i^h, where $B_i^h = \{B_1^h, B_2^h, ..., B_b^h\}$, $\forall i \in B$, $\forall h \in H$. For example, B_3^3 represents the BAN patient 3 that belongs to the hospital 3. For simplicity, it is assumed that the doctors, outpatients and BANs belong to a particular department in the hospital.

In our study, total N number of geo-distributed data centers are considered where both internal and external cloud data centers are included. In our proposed model, an external cloud is adopted for processing of hard deadline based jobs if unable to accommodate in internal cloud which results faster processing and load balancing. In this hybrid model, let m be the number of Inter-cloud Data center (ID) and n be the number of Exter-cloud Data center (ED) are present for healthcare data storage and processing. The ID and ED sets can be represented as $ID = \{ID_1, ID_2, ..., ID_m\}$ and $ED = \{ED_1, ED_2, ..., ED_n\}$, respectively. Hence, $N = \{\{ID_1, ID_2, ..., ID_m\} \cup \{ED_1, ED_2, ..., ED_n\}\}$. Let, ℓ be the number of gateways are connected with h number of hospitals for data transmission and the gateway set can be represented as $G = \{G_1, G_2, ..., G_\ell\}$. Those m number of IDs are connected with ℓ number of user-side gateways and n number of EDs for data transmission. The user generated data and requests are redirected by the gateways G_i to any ID_j or ED_k in multi-cloud, where $i \in G$, $j \in ID$ and $k \in ED$.

3.2. Proposed Spark Architecture

In this subsection, a healthcare big data analytic and processing architecture is proposed for both streaming and batch data using multi-cloud (*ID* and *ED*) Spark platform as shown in Fig. 2. All healthcare data are collected from different users such as doctors, outpatients and BAN patients. Basically, the Request Handler is responsible for interaction and handling of the data and computation intensive queries. Hence, a Request Handler is used in our proposed model for handling data and query. The collected data are categorized as either batch or streaming type based on their arrival rates

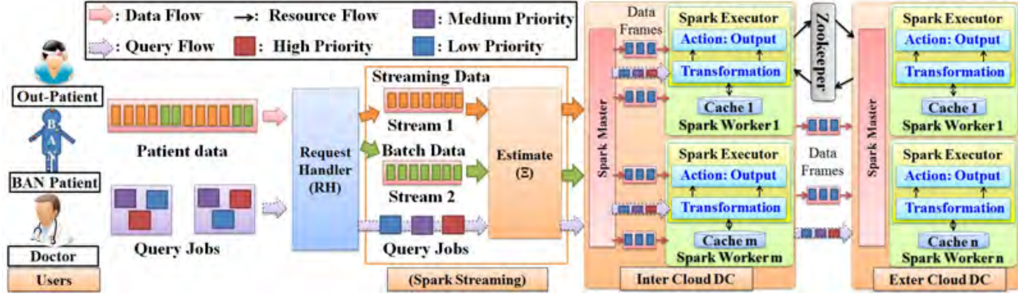

Figure 2: Proposed Spark architecture for streaming and batch data processing.

(λ). In Spark Streaming, the patient data and upcoming analysis requests are divided into small RDDs objects and channeled into different parallel streams based on their λ, block (β_L) and batch (β_A) intervals. Let us consider a cardiac patient and an orthopedic patient, where data of the Cardiac patient are generated much faster as compared to an Orthopedic patient as Cardiac related to the ongoing observation of the hearts i.e. streaming data and Orthopedic is related to the periodic observation or batch data. In this example, streaming and batch data are channelized into stream 1 and 2, respectively. Similarly, the jobs queued into the streams based on the priority of the patients are handled by the Request Handler. Before execution of any job in the IDs or EDs, the Estimator (Ξ) must calculate the required processing time (T^P) and required memory (ξ_R) of the job during the profiling phase.

In this healthcare scenario, some emergency patients exist, where prioritized jobs such as the data analysis of the Intensive Care Unit (ICU) patients and the doctors query during any operation are also executed continuously. Therefore, the emergency patient jobs are prioritized based on the SLA, current health conditions (α), disease severity (ψ) and emergency factor (ε). Hence, the patient's health condition analysis must be finished within the SLA which refers to the hard deadline. Similarly, some batch jobs are executed on the large volume of healthcare data such as any chronic disease patients historical data analysis which can tolerate some admissible time delay known as a soft deadline. In this situation, hard deadline based jobs get higher priority than soft deadline based jobs.

The collected patient data are placed in the local cache memory (ξ_c) of the worker nodes by the master node based on the availability of the storage space. Even, if some data blocks are not fit in ξ_c, then those extra blocks

are transferred to the secondary memory (ξ_s) of the worker within the data center. Similarly, the jobs are also assigned to the Spark worker nodes by the Spark master in the IDs, if the estimated time is within the SLA. Otherwise, the Spark master estimates the T^P and ξ_R of the EDs by considering the data transfer time (T^τ). It is assumed that there is no data transfer time within the IDs. This profiling phase of the job is executed on both batch and streaming type data sets. Zookeeper is used to update and manage the resources between IDs and EDs. Eventually, the jobs are assigned to the most suitable worker nodes by the Spark master which having minimum processing time based on the previous estimation Ξ. Finally, the output of the jobs from IDs and EDs are combined together to produce the final output and send back to the users for better patient care.

4. Healthcare Big Data Computation

Big Data computation is a major aspect for any data intensive applications due to agile and immense data volume. In our Big Data processing, the healthcare patients data and their associated jobs are processed in parallel and distributed fashion on Spark platform. Before any data analysis, the data must be placed on the worker nodes to achieve the data locality, which has great impact on the execution time. To achieve the best locality for both batch and streaming data, an *ODD* algorithm is proposed and discussed in this section. Besides, the analytical jobs must be executed efficiently without violating the *SLA* by considering the resource constrained such as CPU, memory and network bandwidth. Hence, a *DJS* algorithm is explained in this section by considering the resource constrained and priority of the jobs. In patient monitoring, some emergency patients may arrive based on their current health status, disease severity and emergency condition. Hence, a *GPS* algorithm is proposed to prioritize the patient data and jobs based on the health conditions. **The symbols for healthcare big data computation are listed in the Table 1**.

4.1. Global Priority Setting

In this section, the global priority of the patient is decided by considering the *SLA*, current health condition (α), disease severity (ψ) and emergency factor (ε). Let, ρ^{ϕ}_{min} and ρ^{ϕ}_{max} be the minimum and maximum range for each health parameter (ϕ), respectively. **In healthcare domain, doctors rely on the value of clinical outcomes of the health parameters as**

ACCEPTED MANUSCRIPT

Table 1: Symbols and description for healthcare big data computation.

Symbols	Description	Symbols	Description
α	Current health condition	ψ	Disease severity
ε	Emergency factor	ϕ	Health parameter
Δ	Admissible range of each ϕ	ω	Weight
ρ_{min}^{ϕ}	Minimum range of ϕ	f	Priority function
ρ_{max}^{ϕ}	Maximum range of ϕ	γ	Priority coefficients
λ	Arrival rates	Γ	Global priority
β_L	Block interval	β_A	Batch interval
ζ	Number of jobs	δ	Number of data blocks
X	Number of workers assigned for streaming processing	Y	Number of workers assigned for batch processing
ξ	Storage memory	η	Size of each data block
Ξ	Estimator	C	Number of CPU cores
T^{τ}	Transfer time	T^{E}	Job execution time
ϖ	Transferred packet size	∂	Data transmission rate
L_D	Link distance	S_P	Propagation speed
Q_L	Queue length	Λ	Job scheduling function

an evidence of the disease. As investigated in [35], stage IIa in high risk (IIaHR) and low risk (IIaLR) colon cancer patients is identified based on the pathological features. It is observed that maximum diameter of a tumor in IIaLR group is $<$ 4 cm, whereas the tumor diameter in IIaHR group is $>$ 4 cm. Similarly, age, Body Mass Index (BMI), Hemoglobin, etc. can have different level of severity and therefore, we generalize the disease risk of a patient by assigning α, ψ, and ε. For example, *Eosinophil* is a parameter for the patient, i.e. $\phi = Eosinophil$, where the normal range is $\rho_{min}^{Eosinophil} = 0$ to $\rho_{max}^{Eosinophil} = 5$. During the priority setting, α, ψ and ε are set based on the value of ρ_{min}^{ϕ} and ρ_{max}^{ϕ}. If value of ϕ is within the range of ρ_{min}^{ϕ} and ρ_{max}^{ϕ}, α is set as α^N, ψ is set as ψ^L and ε is set as ε^N, where α^N, ψ^L and ε^N are the values of normal condition for α, ψ and ε, respectively. Similarly, in another situation, if value of ϕ lies within $\rho_{min}^{\phi} -\Delta$ and $\rho_{max}^{\phi} +\Delta$ from the normal ranges, then α is set as α^S, ψ is set as ψ^H and ε is set as ε^N, where α^S, ψ^H and ε^N are the serious, high and no emergency condition values for α, ψ and ε, respectively. Here, Δ is the admissible range of each ϕ for each patient P. In case of an emergency condition, α, ψ and ε are set as α^S, ψ^V and ε^E for

the patients, where ψ^V represents very high disease severity and the value of ϕ is lower or higher than $\rho^{\phi}_{min} - \Delta$ and $\rho^{\phi}_{max} + \Delta$ with emergency condition. **If multiple parameters and their impact on disease severity are considered, variance and correlation among the parameters should be calculated for the analysis. Based on the correlation value as High or Low, severity and emergency factors can be assigned. In this paper, the threshold values are generalized as they vary from one disease to another. These threshold values can be set by the analyst by examining the priority of the patient from the clinical outcomes.** Further, a weight (ω) is calculated based on α and ψ in the initial step of *GPS* algorithm as shown in Eq. 1.

$$\omega_i(t) = \alpha_i(t) * \psi_i(t) \tag{1}$$

After determining ω, the global priority function is evaluated to calculate the priority among the patients and their associated jobs (analysis and queries). In this study, the *logistic function* is used as the priority function ($f(\omega_i(t))$) for evaluation as shown in Eq. 2.

$$f(\omega_i(t)) = \frac{SLA_i(t)}{1 + e^{-\omega_i(t)*\varepsilon(t)}} \tag{2}$$

In our analysis, three types of priority coefficients are considered to stabilize the global priority, i.e. High (γ^H), Medium (γ^M) and Low (γ^L) as shown in Eq. 3.

$$\gamma = \begin{cases} \gamma^H & \# \textit{ For High Priority} \\ \gamma^M & \# \textit{ For Medium Priority} \\ \gamma^L & \# \textit{ For Low Priority} \end{cases} \tag{3}$$

Further, the global priority Γ is calculated based on value of γ and $f(\omega)$ as given in Eq. 4.

$$\Gamma(t) = \frac{\gamma(t)}{\sum \gamma(t)} * f(\omega(t)) \tag{4}$$

The step by step procedures of priority calculation are shown in Algorithm 1. Let us consider an example to explain the priority setting algorithm of the jobs among different patients. In this example, let P_1, P_2 and P_3 be the patients having jobs J_1, J_2 and J_3, respectively. Let, the *SLA* values are same for P_1 and P_2. However, it is different for the patient P_3. For example, the

SLA value for P_1 and P_2 is 5 seconds, whereas P_3 has 6 seconds. Similarly, the value of γ^H, γ^M and γ^L are set to be 2, 1 and 0, respectively. According to the priority algorithm, the values of α_i, ψ_i and ε_i are set for each patient i. In this healthcare scenario, there will be three cases of priority such as High, Medium and Low.

4.1.1. Case 1: High Priority

The priority for a patient is considered to be high, if and only if current health condition of a patient is serious and disease severity along with emergency factor is large for which γ is set as γ^H. For example, let P_1 be a serious patient with high value of disease severity and emergency situation such that value of α_1, ψ_1 and ε_1 is 0.2, 0.2 and 0.1, respectively. Hence, the weighted factor ω_1 is calculated based on the above conditions and is found to be 0.04. Therefore, the priority function $f(\omega_1)$ is calculated as 2.50. Eventually, value of Γ_1 is calculated based on Eq. 4 and is found to be 1.66.

4.1.2. Case 2: Medium Priority

A patient's data is considered to be medium priority if current health condition is serious, disease severity is high and emergency factor is small for which γ is set as γ^M. In this case, let P_2 be the patient with high disease severity such that value of α_2, ψ_2 and ε_2 is 0.9, 0.2 and 1, respectively. Hence, the weighted factor ω_2 is calculated based on the above conditions and is found to be 0.18. Therefore, the priority function $f(\omega_2)$ is calculated as 2.73. Eventually, the value of Γ_2 is found to be 0.91.

4.1.3. Case 3: Low Priority

A patient's data is set to be low priority if current health condition is normal, disease severity is low and emergency factor is also small for which γ is set as γ^L. In this case, let P_3 be the patient with normal health condition and low disease severity such that value of α_3, ψ_3 and ε_1 is 0.9, 0.9 and 1, respectively. Hence, the weighted factor ω_3 is calculated as 0.81 and the priority function $f(\omega_3)$ is calculated as 4.16. The value of Γ_3 is found be 0.

It is to be noted that the priorities are arranged in descending order of the values of Γ. In this example, order of the priorities is Γ_1, Γ_2 and Γ_3. It is observed that the priority decreases if value of α, ψ, ε and SLA increases and vice-versa. For example, Γ_3 has a lower priority as values of α, ψ, ε and SLA are higher as compared to the value of Γ_1 and Γ_2. This GPS algorithm can be implemented on all patients and their associated jobs. In case of

ACCEPTED MANUSCRIPT

Algorithm 1 Global Priority Setting (GPS) Algorithm

Require: $\alpha_i(t)$: Current health condition of ith patient at time t.
$\psi_i(t)$: Disease severity of ith patient at time t.
$\varepsilon_i(t)$: Emergency factor of ith patient at time t.
SLA : SLA value (in time) for processing of the jobs.
ϕ_i : be the number of health parameters of ith patient.
ρ^{ϕ}_{min} : Minimum range of ϕ_i.
ρ^{ϕ}_{max} : Maximum range of ϕ_i.
Δ : The admissible range of ϕ_i over the normal range.
Ensure: $\Gamma_i(t)$: The global priority of the patient at time t.
Notations: $\alpha^S_i(t)$: Serious health condition of ith patient at time t.
$\alpha^N_i(t)$: Normal health condition of ith patient at time t.
$\psi^V_i(t)$: Disease severity is very high of ith patient at time t.
$\psi^H_i(t)$: Disease severity is high of ith patient at time t.
$\psi^L_i(t)$: Disease severity is low of ith patient at time t.

1: Initialize $\omega^p(t) = 0$;
2: $\Gamma^p = \{\}$;
3: **for** each patient p in P **do**
4:     **if** $\rho^{\phi}_{min} \leq \phi_i \leq \rho^{\phi}_{max}$ **then**
5:         $\alpha^N = \alpha_i$, $\psi^L = \psi_i$ and $\varepsilon^N = \varepsilon_i$;
6:     **else if** $\rho^{\phi}_{min} - \Delta \leq \phi_i \leq \rho^{\phi}_{max} + \Delta$ **then**
7:         $\alpha^M = \alpha_i$, $\psi^M = \psi_i$ and $\varepsilon^N = \varepsilon_i$;
8:     **else**
9:         $\alpha^H = \alpha_i$, $\psi^H = \psi_i$ and $\varepsilon^E = \varepsilon_i$;
10:     **end if**
11:     The weights (ω) is calculated based on Eq. 1;
12:     The priority function $(f(\omega))$ is evaluated based on Eq. 2;
13:     **if** $\alpha_i(t) = \alpha^S_i(t)$ && $\psi_i(t) = \psi^V_i(t)$ && $\varepsilon_i(t) = \varepsilon^E$ **then**
14:         Set γ as γ^H based on Eq. 3 ;
15:     **else if** $\alpha_i(t) = \alpha^S_i(t)$ && $\psi_i(t) = \psi^H_i(t)$ && $\varepsilon_i(t) = \varepsilon^N$ **then**
16:         Set γ as γ^M based on Eq. 3 ;
17:     **else**
18:         Set γ as γ^L based on Eq. 3 ;
19:     **end if**
20:     Calculate the global priority $\Gamma_i(t)$ based on Eq. 4;
21:     Arrange the patients in descending order of $\Gamma_i(t)$;
22: **end for**
23: Return $\Gamma_i(t)$;

ACCEPTED MANUSCRIPT

same priority label for multiple patients is found, a random selection can be performed . Hence, the data and jobs are queued based on the global priority of the patients for scheduling and processing without violating the *SLA*.

4.2. Optimal Data Distribution

The main objective of optimal data distribution mechanism is to allocate both batch and streaming data blocks efficiently among the Spark worker nodes. **Spark streaming supports mini batches of data sets as input. Based on the arrival rate, we divide the input data into batches and streaming jobs, which is supported by Spark DStream, a sequence of RDDs [36].**

It is assumed that the patient data are alighted with different arrival rates (λ). Initially, λ_i is checked for each i-th patient's incoming record. If the patient data arrival rate is continuous and high (For example, 10pkts/sec) with less packet size (For example, 10mb/pkt), those data are treated as the streaming data sets. The streaming data arrival rate can be symbolized as λ_i^S for i^{th} patient. Likewise, if the data arrival rate is low (For example, 5pkts/sec) with large packet size (For example, 100mb/pkt), those data are treated as the batch data sets. The batch data arrival rate can be symbolized as λ_j^B for j^{th} patient. By taking the advantages of Spark platform, the streaming and batch data are dispatched in parallel by using multiple streams to the Spark workers. Once the input data are segregated as batch and streaming data sets, the Block interval (β_L) and Batch interval (β_A) are set for each type of patient data. By adjusting β_L and β_A, the total number of jobs (ζ) can be calculated to know the overall throughput of the system. The definition of β_L, β_A and ζ are described as follows.

Definition 1. ***Block interval (β_L):*** *The block interval is defined as the interval at which the data are received by the Spark streaming receivers and are chunked into blocks of data before storing in the Spark nodes.*

Definition 2. ***Batch interval (β_A):*** *The batch interval is defined as the interval in which mini-batches are received by the Spark master, where mini-batches are the combination of multiple data blocks.*

Definition 3. ***Total # of jobs (ζ):*** *The total number of jobs per stream per batch can be defined as the ratio of the batch interval (β_A) and block interval (β_L).*

ACCEPTED MANUSCRIPT

In the Spark model, the number of blocks in each mini-batch determines the number of ongoing jobs to execute on the worker nodes. Let, β_L^S and β_A^S be the block and batch interval of the streaming data, respectively. For example, β_L^S and β_A^S are set to be 100ms and 1 second, respectively. Let, ζ^S be the total number of jobs that can be executed on the streaming data sets. Hence, ζ^S can be evaluated as $\frac{\beta_A^S}{\beta_L^S}$. In this example, ζ^S can be executed as 1000ms/100ms, which is 10. Similarly, let β_L^B and β_A^B be the block and batch interval for batch type data sets, respectively. For example, β_L^B and β_A^B are set to be 1000ms and 5 seconds, respectively. Let, ζ^B be the total number of jobs that can be executed on batch type data sets. Hence, ζ^B can be calculated as $\frac{\beta_A^B}{\beta_L^B}$. In this example, ζ^B can be executed as 5000ms/1000ms, which is 5. If the numbers of jobs are less than the numbers of cores per CPU within a Spark worker node, an underload situation occurs. To balance this situation, the number of jobs can be increased by reducing the block intervals of each job for the batch intervals. **It is to be noted that the data size is same within the mini-batches (batch intervals). However, it can be changed in the next batch intervals based on each block size and incoming job size. The computation of the assigned jobs is same within the batch interval β_A. However, it can be changed in the next interval based on the availability of resources such as CPU and memory.**

Data placement is a major determinant of the performance of Spark jobs. For that reason, the data blocks are placed in such a way that most of the jobs can achieve data locality during the execution. In our work, an Optimal Data Distribution (ODD) algorithm is proposed for better data locality and faster execution of the jobs. The complete steps of our proposed *ODD* algorithm are shown in Algorithm 2. In our Spark model, the workers are divided into two different groups within the data centers. Let X and Y be the number of workers assigned for streaming and batch data processing, respectively. The basic difference between the streaming and batch worker is the processing capability and memory size. The streaming workers are having less ξ_c and high processing capability whereas the batch workers are having high ξ_c than the streaming workers. Let, $\delta(t)$ be the total number of data blocks coming from the set $\Re$ after applying *MCA* algorithm at time t. Moreover, out of those $\delta(t)$ blocks, δ^S and δ^B number of streaming and batch data blocks need to be placed on the respective workers for better performance in terms of less T^P, where $\delta(t) = \delta^S(t) + \delta^B(t)$. Besides, each β_A^S and β_A^B must fit in the

ACCEPTED MANUSCRIPT

Algorithm 2 Optimal Data Distribution (ODD) Algorithm

Require: $\delta(t)$: Total number of data blocks coming at time t.
$\lambda_i(t)$: Data arrival rate for i^{th} patient at time t.
Ensure: Optimal placement of each data block δ at time t.
Notations:
$\lambda_i^S(t)$: Streaming data arrival rate for i^{th} patient at time t.
$\lambda_i^B(t)$: Batch data arrival rate for i^{th} patient at time t.
ξ_R^{SL} and ξ_R^{SB} : Required memory of streaming and batch data.
X : Set of workers assigned for streaming job processing.
Y : Set of workers assigned for batch job processing.

1: $\delta(t) = \delta^S(\text{t}) + \delta^B(\text{t})$;
2: **for** each data block δ_i **do**
3:   **if** λ_i = "HIGH" && "CONTINUOUS" **then**
4:     $\delta^S = \text{i}$; # For streaming data.
5:   **else**
6:     $\delta^B = \text{i}$; # For batch data.
7:   **end if**
8:   Calculate β_L^S, β_L^B, β_A^S, and β_A^B intervals;
9:   Evaluate $\zeta^S = \frac{\beta_A^S}{\beta_L^S}$;
10:   Evaluate $\zeta^B = \frac{\beta_A^B}{\beta_L^B}$;
11:   Separate X for streaming and Y for batch data processing nodes;
12:   Calculate ξ_R^{SL} and ξ_R^{SB} based on Eq. 5;
13:   **if** $\xi_R^{SL} \leq \xi_c^S$ **then**
14:     Assign δ_i^S into ξ_c^S of streaming worker $[i] \in X$;
15:   **else**
16:     Assign δ_i^S into ξ_s^S of streaming worker $[i] \in X$;
17:   **end if**
18:   **if** $\xi_R^{BL} \leq \xi_c^B$ **then**
19:     Assign δ_i^B into ξ_c^B of batch worker $[i] \in Y$;
20:   **else**
21:     Assign δ_i^B into ξ_s^B of batch worker $[i] \in Y$;
22:   **end if**
23: **end for**

ACCEPTED MANUSCRIPT

streaming (ξ_c^S) and batch (ξ_c^B) local cache, respectively. The required cache memory is calculated based on the input data blocks as given in Eq. 5 to store the data sets.

$$\left.\begin{array}{r}\xi_R^S(t) = \delta^S * \eta^S \\ \xi_R^B(t) = \delta^B * \eta^B\end{array}\right\} \tag{5}$$

Where, ξ_R^S and ξ_R^B are the total required memory for storage of incoming streaming and batch data, respectively. In Eq. 5, η^S and η^B are the size of each streaming and batch data blocks, respectively.

The required memories, ξ_R^S and ξ_R^B must fit in ξ_c^S and ξ_c^B, respectively. However, if ξ_R^S and ξ_R^B are not fit in ξ_c^S and ξ_c^B, then the remaining data blocks are sent to the streaming (ξ_s^S) and batch (ξ_s^B) secondary memory of the worker nodes. In another scenario, if the data blocks are large in volume and cannot be accommodated in a single worker node, then the remaining blocks are placed in another local worker node. Basically, the received RDDs are automatically cleared after execution in Spark streaming. However, the persisted RDDs are used to store the RDDs in local memory for future use and accessible to outside the streaming application.

4.3. Dynamic Job Scheduling

In this section, a Dynamic Job Scheduling (DJS) mechanism is proposed for Apache Spark platform to minimize the processing time without violating *SLA*. To the best of our knowledge, priority based patient job scheduling in Spark platform is the first work in healthcare domain. Prior to the scheduling, *GPS* algorithm is executed to prioritize the patient and their related jobs. The data related to the prioritized patients are placed onto the worker nodes by using our proposed *ODD* algorithm. Hence, the *DJS* algorithm is applied on those prioritized jobs for placement. Specifically, a profiling based analytic model is designed to minimize the job completion time. **Here, the profiling phase is offline before the real execution. This ensures the reduction of waiting time of the jobs. Once a job is submitted to the Spark framework, the Estimator (Ξ) receives the profiling information and uses the DJS and analytic model for the processing purpose. First of all, the near optimal processing time and required memory are calculated by the estimator for the sample jobs in the profiling phase.** Let, Ξ_{CPU} and Ξ_ξ be the processing and memory estimator for the sample jobs, respectively. Later stage of the *DJS* algorithm, all

ACCEPTED MANUSCRIPT

other jobs are scheduled and placed on the Spark worker nodes by comparing the profiling results. Since, Streaming and Batch jobs are considered in our analysis, profiling is done differently based on the types of jobs. Irrespective of the job types, the estimator Ξ_{CPUij} and Ξ_{CPUij} can be expressed as given in Eq. 6, where the processing time estimator of ith job is scheduled either in j^{th} worker node of IDs or k^{th} worker node of EDs, where $j \in X^{ID}$ and $k \in X^{ED}$.

$$\left.\begin{aligned} \Xi_{CPUij} &= T^{E}_{ij}, i \in \zeta, j \in X^{ID}, \text{ \# Data in IDs} \\ \Xi_{CPUik} &= T^{E}_{ik} + T^{\tau}_{ik}, i \in \zeta, k \in X^{ED}, \text{ \# Data in EDs} \end{aligned}\right\} \tag{6}$$

Where, T^{E}_{ij} is the ith job execution time on j^{th} worker node in IDs when data locality is achieved. Similarly, T^{E}_{ik} and T^{τ}_{ik} are the execution and transfer time, respectively on k^{th} worker node in EDs when the data locality is not achieved within the IDs or the estimated execution time is greater than the SLA, i.e. $T^{E}_{ij} > SLA_i$. T^{E}_{ij} and T^{E}_{ik} are calculated as deduced in Eq. 7.

$$\left.\begin{aligned} T^{E}_{ij} &= \frac{(\zeta_T - \zeta_E) * T^{\zeta}_{i}}{\beta_{Ai} * C_j} \\ T^{E}_{ik} &= \frac{(\zeta_T - \zeta_E) * T^{\zeta}_{i}}{\beta_{Ai} * C_k} \end{aligned}\right\} \tag{7}$$

Where, ζ_T and ζ_E are the total and already executed streaming jobs, respectively. **Here, T^{ζ}_{i} is the execution time, which is defined as the time taken by the ith job to be processed in a specific worker node.** β_{Ai} is the batch interval of ith job. In this equation, C_j and C_k are the number of CPU cores present in j^{th} and k^{th} node of ID and ED, respectively. Conversely, if the node cannot achieve data locality in the IDs or the T^{E}_{ij} is higher than the SLA, then the job must be moved to the EDs for execution. Hence, the additional network traffic delay is occurred. The additional traffic delay T^{τ}_{ik} for i^{th} jobs can be expressed as given in Eq. 8.

$$T^{\tau}_{ik} = \frac{\varpi}{\partial} + \frac{L_{Dk}}{S_{Pk}} + \frac{Q_{Lk}}{\lambda_i} \tag{8}$$

Where, ϖ is the transferred packet size, ∂ is the data transmission rate and λ_i is the data arrival rate of ith job. L_{Dk}, S_{Pk} and Q_{Lk} are the link distance, propagation speed and queue length of the network, respectively for the worker node $k \in X^{ED}$. Further, the estimator for the required cache

ACCEPTED MANUSCRIPT

memory $\Xi_{\xi i}$ of ith job is estimated as expressed in Eq. 9, which is same for both IDs and EDs.

$$\Xi_{\xi i} = \delta_i * \eta \tag{9}$$

Where, δ_i is the numbers of data blocks needed to be processed for i^{th} jobs and η is the size of each data block.

Upon determining the values of Ξ_{CPU} and Ξ_{ξ} in the profiling phase, the final job assignment is performed based on the values of the profiling parameters. In our analysis, ζ is channelized to the workers by the Request Handler. $\Lambda(\zeta)$ is the job scheduling function for the prioritized jobs as defined in Eq. 10.

$$\Lambda(\zeta_i) = \begin{cases} Assigned\ to\ node\ X_j^{ID} \leftarrow \zeta_i \\ \ If\ (\Xi_{CPUij}\ \leq\ SLA_i\ \&\&\ \Xi_{\xi i}\ \leq\ \xi_{cj}) \\ Assigned\ to\ node\ X_k^{ED} \leftarrow \zeta_i \\ \ If\ (T_{ij}^{E} >\ SLA_i\ \&\&\ \Xi_{CPUik}\ \leq\ SLA_i \\ \qquad \&\&\ \Xi_{\xi i}\ \leq\ \xi_{ck}) \\ Reject \quad Otherwise \end{cases} \tag{10}$$

Where, i^{th} job ζ_i needs to be placed in j^{th} Spark worker node in the IDs, i.e. X_j^{ID}. Here, Ξ_{CPUij} is the estimation time for processing i^{th} job in j^{th} Spark worker, which needs to be less than or equal to the SLA_i. Further, the required memory $\Xi_{\xi i}$ for i^{th} job must satisfy the local cache ξ_{cj} of jth Spark worker node. Hence, the ith job is placed in j^{th} Spark worker if and only if it satisfies both CPU and memory requirements. However, if the T_{ij}^{E} is greater than the SLA_i due to heavy load in node j, the ith job must be executed in EDs to satisfy the SLA_i. Therefore, T_i^P is increased due to occurrence of the data transfer time (T_{ik}^{τ}) from k^{th} node. Eventually, the ith job is scheduled in k^{th} Spark worker of ED, if the processing time and memory satisfy the SLA. The complete steps of DJS is presented in Algorithm 3.

5. Healthcare Big Data Analysis

In healthcare Big Data analysis, we intend to predict the future disease of the patient based on their health parameters. In a medical environment, the immense volume of patient physiological data are generated with high dimensions. In fact, out of all the data sets, only few dimensions of data have very high impact on the disease prediction. Hence, a dimension reduction

ACCEPTED MANUSCRIPT

Algorithm 3 Dynamic Job Scheduling (DJS) Algorithm

Require: SLA : SLA value (in time) for processing of the jobs.
Ensure: Schedule each job ζ_i dynamically on Spark workers.

1: **for** for each patient p in P **do**
2: $\Gamma(t)$: Execute GPS();
3: Execute ODD();
4: Arrange the patients in descending order of $\Gamma_i(t)$;
5: **end for**
6: **for** each job ζ_i **do**
7: Estimate Ξ_{CPUi} for i^{th} job as in Eq. 6;
8: Estimate $\Xi_{\xi i}$ for i^{th} job as shown in Eq. 9;
9: **end for**
10: **if** $\Xi_{CPUqi} \leq SLA_i$ && $\Xi_{\xi q} \leq \xi_{ci}$ **then**
11: Assign $\Lambda(\zeta_i)$ to ID worker node X_q^{ID} as in Eq. 10;
12: **else if** $T_i^E > SLA_i$ && $\Xi_{CPUik} \leq SLA_i$ && $\Xi_{\xi k} \leq \xi_{ci}$ **then**
13: Assign $\Lambda(\zeta_i)$ to ED worker node X_k^{ED} as in Eq. 10;
14: **else**
15: **Reject**;
16: **end if**
17: Return $\Lambda(\zeta)$;

ACCEPTED MANUSCRIPT

Table 2: Symbols and description for healthcare big data analysis.

Symbols	Description	Symbols	Description
p	Number of patients	Υ	Class label of the disease
M^{Φ}	Health parameter matrix	M^{CL}	Class matrix
$Count$	Counter variable of the class	υ	Class label
Ω	Threshold	$\Re$	Reduced set of the parameters
$\aleph$	Tuple	Θ	Class probability
μ	Mean value	σ^2	Variance
$\wp$	Conditional probability	E	Evidence

is awfully essential to minimize the data volume and maximize the accuracy of the outcomes. For example, let us consider a patient that belongs to the Cardiology department, where the disease severity is different for each individual with the same number of health parameters. Therefore, in our proposed work, we are interested to find the most influential parameters with respect to the disease within a specific department. Hence, those most influential health parameters with respect to the disease are selected based on our proposed *MCA* dimension reduction algorithm. Later, the future disease of the patients are predicted by using our *PSNB* prediction algorithm based on those influential parameters instead of considering all collected health parameters for analysis. Moreover, dimension reduction and disease prediction are two major components in our proposed healthcare Big Data analysis. **The symbols for healthcare big data analysis are listed in the Table 2**.

5.1. Dimension Reduction

Prior to the data analysis, the high dimensional input data are reduced to a finite set by using dimension reduction mechanism [37]. In our healthcare environment, p number patients present in a hospital, where each patient belongs to a specific department $\forall p \in P$. Each patient is associated with a class label with respect to the disease (P_p, Υ_υ). Let, Υ be the class label set, where υ number of class labels present, i.e. $\Upsilon_i = \{\Upsilon_1, \Upsilon_2, ..., \Upsilon_\upsilon\}$. For example, Υ^{Card} be the class for heart patients, where c can be a heart disease *Yes* or *No*, i.e. $\Upsilon^{Card} = \{\Upsilon_{Yes}, \Upsilon_{No}\}$. Here, *num* represents the class label for heart patient as shown in Fig. 3 *Input Data*, where 0 represents no heart disease, 1, 2 and 3 represent the heart disease with different severity. Let, ϕ be the numbers of health parameters present in a set Φ for each patient P_i, where $\Phi_i = \{\Phi_1, \Phi_2, ..., \Phi_\phi\}$. Before predictive analysis, the health parameters (Φ)

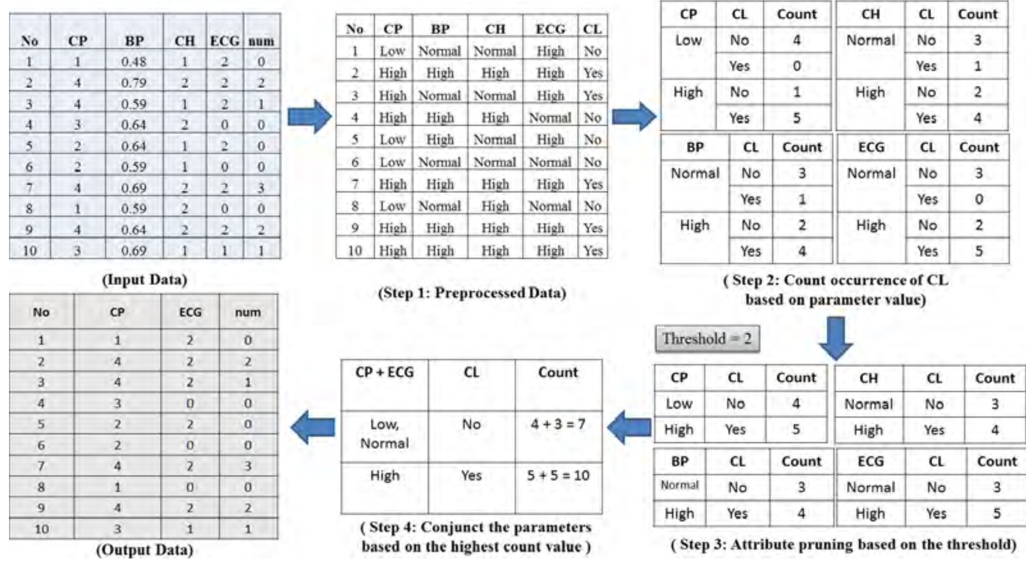

Figure 3: Example of execution of MCA algorithm.

of pth patient are presented in the form of a *health parameter matrix* (M_p^{Φ}) as given in Eq. (11).

$$M_p^{\Phi} = \begin{array}{c} \\ P_p \end{array} \begin{array}{c} \begin{matrix} \Phi_1 & \Phi_2 & . & \Phi_\phi \end{matrix} \\ \begin{pmatrix} (P_p,\Phi_1) & (P_p,\Phi_2) & . & (P_p,\Phi_\phi) \end{pmatrix} \end{array} \tag{11}$$

Besides, total p numbers of patients are available, where $p \in P$ with ϕ number of health parameters. The health parameters are stored in *class matrix* (M_p^{CL}) as given in Eq. (12).

$$M_p^{CL} = \begin{array}{c} \\ P_1 \\ P_2 \\ . \\ P_p \end{array} \begin{array}{c} \begin{matrix} \Phi_1 & \Phi_2 & . & \Phi_\phi \end{matrix} \\ \begin{pmatrix} (P_1,\Phi_1) & (P_1,\Phi_2) & . & (P_1,\Phi_\phi) \\ (P_2,\Phi_1) & (P_2,\Phi_2) & . & (P_2,\Phi_\phi) \\ . & . & . & . \\ (P_p,\Phi_1) & (P_p,\Phi_2) & . & (P_p,\Phi_\phi) \end{pmatrix} \end{array} \tag{12}$$

Let us consider an example, where a patient from the *Cardiology(Crd)* department has 14 different parameters related to the heart disease [38]. For example, p number of *Cardiology* patients have multiple heart disease parameters. Now, the *class matrix* $M_p^{CL_{Crd}}$ can be represented as given below.

ACCEPTED MANUSCRIPT

$$M_p^{CL_{Crd}} = \begin{array}{c} \\ P_1 \\ P_2 \\ . \\ P_p \end{array} \begin{array}{c} \begin{array}{cccccc} Age & Sex & cp & trestbps & . & thalach \end{array} \\ \begin{pmatrix} 62 & 1 & 1 & 145 & . & 145 \\ 68 & 1 & 4 & 132 & . & 160 \\ . & . & . & . & . & . \\ 70 & 2 & 2 & 156 & . & 139 \end{pmatrix} \end{array}$$

In this example, 1 and 0 in Sex column represent the male and female patients, respectively. Similarly, *cp* type can be represented as 1: typical angina, 2: atypical angina, 3: non-anginal pain and 4: asymptomatic pain and other health parameters hold their respective values. It is very tedious task to consider all ϕ number of parameters for a heart patient, where some of them are irrelevant to the disease. Hence, we remove some parameters considering only the most influential parameters using our *MCA* algorithm. **MCA algorithm is applied only once in the initial phase of the analysis as the most influential parameters are fixed for a specific department. However, it is re-executed for analysis once a new parameter is added to the list.** *MCA* algorithm has four identical steps as shown in Fig. 3, i.e. Preprocessing, count the occurrence of class label, attribute pruning based on the threshold and conjunct the parameters based on the highest count value.

Initially, all ϕ number of health parameter attributes are taken as input for our proposed *MCA* algorithm, where $\forall \phi \in \Phi$. In the preprocessing phase, all ϕ number of original attribute values are transformed into an annotated format to represent the severity of that particular health parameter. For example, the *CP* value *1* is transformed to *Low* for patient number *1* in the first step of the *MCA* algorithm as shown in Fig. 3. In the next step of *MCA* algorithm, the class label occurrence for each attribute is counted based on the annotated format values. Let, *Count* be the counter variable to count the class label υ occurrence for each parameter ϕ, where $\forall \upsilon \in \Upsilon$ and $\forall \phi \in \Phi$ as shown in Eq. (13).

$$Count_{\upsilon}^{\phi} = \sum_{i=1}^{p} \sum_{j=1}^{\phi} \sum_{k=1}^{\upsilon} count[M^{CL}[i][j][k]] \tag{13}$$

For example, the *Count* value is *4* for class label *No* of *Low* cheat pain parameter. In the third step of *MCA* algorithm, attribute pruning is performed based on a threshold (Ω) value, which is decided by the lower frequency value. For example, the threshold value is set as *2* in this heart disease case.

ACCEPTED MANUSCRIPT

The parameters are deleted whose *Count* values are less than the threshold. For example, the *Count* value is zero for *Yes* label of *Low* chest pain, which is deleted from the list. Further, the conjunction is carried out for all possible combinations and compute the joint *Count* value. The highest *Count* value is considered as the most influential parameters with respect to the disease. The most influential parameters are selected for the prediction purpose by discarding all other health parameters as they have less influence on the occurrence of the disease. For example, CP and ECG jointly provide the *Count* value as *7* for *No* class label and *10* for *Yes* class label, which is highest among all other joint parameters. Hence, in this example, CP and ECG are considered as the input for predictive analysis. Finally, the most influential parameters are stored in a reduced set $\Re$, where $\Re < M^{CL}$, which is the output of our *MCA* algorithm by which we can reduce the dimension of the health parameters. The formal steps of the *MCA* algorithm is described in Algorithm 4.

5.2. Prediction of Heart Disease

Recently, most of the data analysis methods lack adequate functionality to predict the future health condition of a patient accurately as the healthcare data have many conditional dependencies and uncertainty. Hence, a probabilistic predictive model is designed here to predict the future disease condition of the patients. Basically, the predictive modeling can be Supervised, Unsupervised or Semi-supervised, where the class label is known for Supervised learning. Similarly, the class label is unknown for Unsupervised learning. In some cases, the class label is known for a specific data set, i.e. training data and unknown for other data sets, which are Semi-supervised learning. In our analysis semi-supervised learning is used as the class label, which is known for some existing patients and is unknown for the new patients, which needs an accurate prediction.

The Naive Bayes model has popularity due to its accuracy and efficiency as compared to other state-of-art algorithms. In order to improve the accuracy and make the model more realistic for healthcare, Semi-Naive Bayes model is adopted to overcome the problems of the Naive Bayes model. The Semi-Naive Bayes model allows certain degree of dependency among the input parameters and has different distribution mechanism for future estimation of the parameters. Basically, Gaussian distribution is used for continuous data, where the continuous incoming values (patients) are associated with

ACCEPTED MANUSCRIPT

Algorithm 4 MCA Algorithm

Require: M^{CL} : The health parameters class matrix with the class label.
Ensure: $\Re$: Reduced parameter set.

Notations:

1: Ω = Threshold;
2: $\Psi[] = \{\Psi_1, \Psi_2, ..., \Psi_\psi\}$;
3: $\Phi[] = \{\Phi_1, \Phi_2, ..., \Phi_\phi\}$;
4: $\Upsilon[] = \{\Upsilon_1, \Upsilon_2, ..., \Upsilon_\upsilon\}$;
5: Count[i][j][k] = 0;
6: Transform each M_{ij} into an annotated format based on Class Label and severity;
7: **for** each patient p in P **do**
8:   **for** each parameter ϕ in Φ **do**
9:     Calculate the $Count_\upsilon^\phi$ value for each class label based on Eq. (13);
10:   **end for**
11: **end for**
12: **if** Count[p][ϕ][υ] $< \Omega$ **then**
13:   Discard Φ_ϕ;
14: **else if** Count[p][ϕ_i, ϕ_j][υ] $< \Omega$ **then**
15:   Discard Φ_{ϕ_i,ϕ_j}, where $i \neq j$;
16:   Otherwise, $\Re = \{\Phi_{\phi_i,\phi_j}\}$;
17: **end if**
18: Return $\Re$;

ACCEPTED MANUSCRIPT

each class (Heart disease class) in a Gaussian manner. Therefore, Gaussian distribution is used in our healthcare scenario for Semi-Naive Bayes.

Let, a *Tuple* ($\aleph$) exists for each patient such that $\aleph = \langle P, \Upsilon \rangle$, where a class label Υ_v is assigned to each patient P_p, i.e. $\forall v \in \Upsilon$ and $\forall p \in P$. Further, the class probability ($\Theta(\Upsilon|\phi)$) is predicted for any unlabeled patient p. $\Theta(\Upsilon|\phi)$ is evaluated by calculating different sub-steps such as mean (μ_ϕ), variance (σ_ϕ^2) and conditional probability ($\wp$). Further, μ_ϕ^v is calculated for each class label Υ_v, where $\forall v \in \Upsilon$, which is expressed in Eq. (14).

$$\mu_\phi^v = \frac{1}{\phi p} \sum_{\imath=1}^{\phi} \sum_{\jmath=1}^{p} M_p^{CL}[\imath][\jmath] \tag{14}$$

After the mean calculation, the variance (σ_ϕ^2) is evaluated for entire cluster matrix as expressed in Eq. (15).

$$\sigma_\phi^2 = \frac{1}{p} \sum_{\jmath=1}^{\phi} \sum_{\imath=1}^{p} (P_\imath^v \phi_\jmath - \mu_\phi^v)^2 \tag{15}$$

Further, the conditional probability for each class with respect to the parameters ($\wp(v|\phi)$) is calculated using Gaussian Semi-Naive Bayes model as shown in Eq. (16).

$$\wp(v|\phi) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{\left(\frac{-(\phi_p^v - \mu_\phi^v)^2}{2\sigma^2}\right)} \tag{16}$$

where, ϕ_p^v is the input value for the attribute ϕ of the class v that belongs to p^{th} patient. Furthermore, the probability ($\Theta(v|\phi)$) of the class (v) for the patient p is calculated using equation Eq. (17).

$$\Theta(v|\phi) = \arg\max_{v_i} \left(\wp(v_i) \prod_{j=1}^{\phi} \frac{\wp(v_i|\phi_i, \phi_j)}{E} \right) \tag{17}$$

Where E is the evidence described in Eq. (18).

$$E = \sum_{i=1}^{v} \prod_{j=1}^{\phi} \wp(v_i|\phi_j), \ where \ i \neq j. \tag{18}$$

Let us continue the example as discussed in dimension reduction section. After dimension reduction, the reduced parameters are kept in the reduced

ACCEPTED MANUSCRIPT

set $\Re$. In our example, *CP* and *ECG* are two input attributes present in $\Re$ as our training set, where $\Re = \{CP, ECG\}$. Let, Υ^{Card} be the heart disease class for heart patients, where $\Upsilon^{Card} = \{\upsilon_{Yes}, \upsilon_{No}\}$. Further, the mean value is calculated separately for all *Yes* and *No* patients i.e. μ^{Yes}_{CP}, μ^{Yes}_{ECG}, μ^{No}_{CP} and μ^{No}_{ECG}. Afterward, the variance is evaluated for both *CP* and *ECG* parameter of the class *Yes* and *No*, i.e. $\sigma^{Yes^2}_{CP}$, $\sigma^{Yes^2}_{ECG}$, $\sigma^{No^2}_{CP}$ and $\sigma^{No^2}_{ECG}$. Further, the conditional probability ($\wp$) is calculated for individual and conjuncted *CP*, *ECG* of *Yes* and *No* class, i.e. $\wp(Yes|CP)$, $\wp(Yes|ECG)$, $\wp(No|CP)$, $\wp(No|ECG)$, $\wp(Yes|(CP, ECG))$ and $\wp(No|(CP, ECG))$. Eventually, $\Theta(Yes)$ and $\Theta(No)$ probabilities are calculated for each patient in the training phase. Finally, testing is performed to predict the class label of an unknown patient, i.e. Υ_u, where *CP* and *ECG* values are *4* and *2*, respectively. $\Theta(Yes_u)$ and $\Theta(No_u)$ are calculated and the class is assigned based on the maximum value. Here, value of Θ can be negative, greater than or less than 1 as probability density is used rather than a probability for continuous values of CP and ECG. The value of $\Theta(Yes_u)$ is larger than $\Theta(No_u)$ as shown in Fig. 4. Hence, the patient having heart disease for respective *CP* and *ECG* values can have necessary medications as prescribed by the doctors.

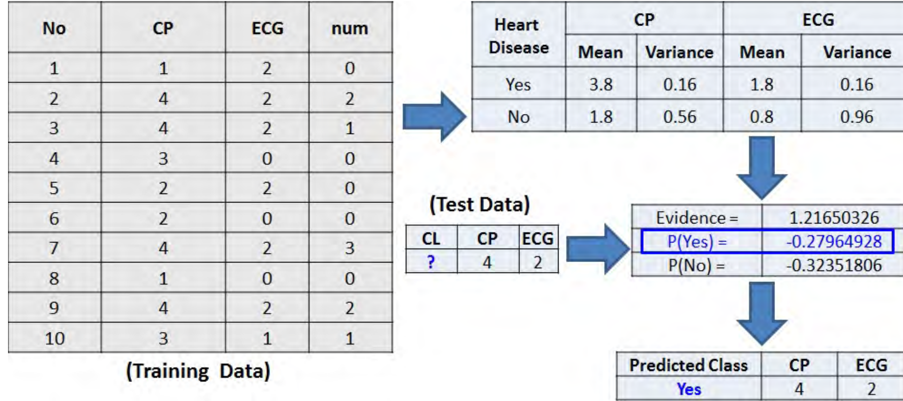

Figure 4: Prediction Example

The *SNB* is executed on the Spark Platform in a parallel and distributed manner. **The *PSNB* algorithm is designed to improve the accuracy and reduce the execution time by taking the advantages of both Bayesian network and Spark parallelism. In the first stage of the PSNB algorithm, the dependant and independent Direct Acyclic Graph (DAG) are created based on the Bayesian influence**

ACCEPTED MANUSCRIPT

Algorithm 5 PSNB Algorithm

Require: $\Re$: Reduced parameter set.
Ensure: $\Theta(\upsilon|\phi)$: Predicted class probability.

1: ssc = spark.StreamingContext();
2: **for** each patient p in P **do**
3:    **for** each parameter ϕ in $\Re$ **do**
4:       Design the dependant and independent DAGs;
5:       Load RDDs = spark.read.load(DAGs);
6:       RDDData = ssc.parallelize(RDDs);
7:       flatmapOp = RDDData.flatmap{
8:         Calculate the mean (μ_ϕ) using Eq. (14);
9:         Calculate variance (σ_ϕ^2) using Eq. (15);
10:         Calculate conditional probability ($\wp$) using Eq. (16);
11:       }
12:       predictionOp = map(flatmapOp){
13:         Calculate the class probability ($\Theta(\upsilon|\phi)$) using Eq. (17);
14:       }
15:    **end for**
16: **end for**
17: action = spark.write.save($\Theta(\upsilon|\phi)$);
18: Return action and $\Theta(\upsilon|\phi)$;

network. Since, some influential parameters exist in the system, an influencal DAG is created. Similarly, all other DAGs are created based on the dependant and independent parameters. In the next stage, the DAGs are assigned to the Spark RDD objects for parallel execution. In the next stage, the mean (μ_ϕ), variance (σ_ϕ^2) and the conditional probability ($\wp$) are executed in the *Transformation: flatmap* function of the *PSNB* algorithm. In the subsequent step of *PSNB* algorithm, the class probability ($\Theta(v|\phi)$) is predicted using the *Transformation: map* function of Spark by using previous output of μ_ϕ, σ_ϕ^2 and $\wp$. Finally, the prediction output is stored using *Action: Save* function for the users in the *PSNB* algorithm. Both *flatmap*, and *map* jobs are executed on appropriate Spark worker nodes based on the DJS algorithm for each prioritized patient. The formal steps of *PSNB* algorithm is described in Algorithm 5.

The time complexity of Naive Bayes (NB) algorithm is $O(pR)$, where p is the number of patients and R is the number of reduced parameters present in the reduced set $\Re$. The time complexity of the MCA algorithm is $O(p\phi)$, where ϕ is the number of health parameters for each patient. Hence, the total time complexity of our proposed PSNB algorithm is $O(pR + p\phi)$.

6. Performance evaluation

Performance of our proposed algorithms is evaluated in a Spark cluster that comprises one Master and 10 Worker nodes. The simulation results are compared with the existing methods. Out of 10 Worker nodes, 5 nodes are used in the IDs and another 5 nodes are used in the EDs to reveal the performance in the real-world scenarios. The Spark Master is installed in Asus Rack server (RS700-X7/PS4) with Ubuntu 14.04 LTS Operating System, Intel Xeon(R) CPU ES-2620v2 2.10GHz x 12 CPU, 16GB memory and 1 TB storage configuration. All workers are installed in commodity hardware with Ubuntu 14.04 LTS Operating System, Intel Core i7-6700 CPU 3.40GHz x 8 CPU, 8GB memory and 1 TB storage configuration. Apache Ambari is used as the data platform management tool for the cluster provisioning, maintenance and management irrespective of the clusters. The major components of Apache Ambari are Hadoop 2.7.1, Spark 1.5.2., HDFS, YARN 2.7 and ZooKeeper 2.3.4, Scala 2.10.4. In our experimental setup, two nodes are considered in ID and another two nodes are used for implementing the ED.

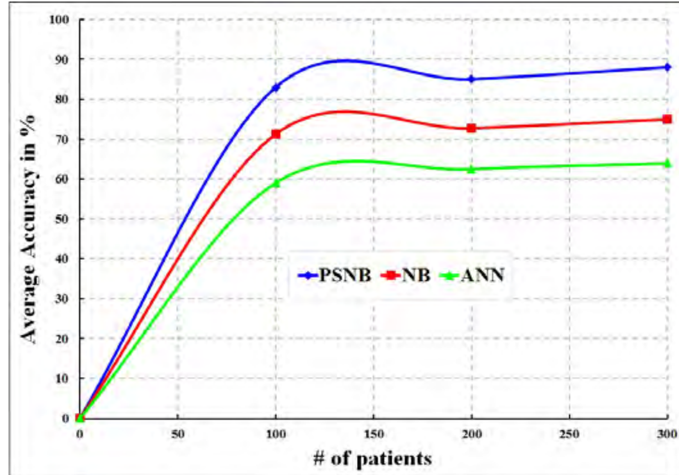

Figure 5: Average prediction accuracy.

To manifest the prediction accuracy of our proposed PSNB algorithm, experiments are performed and compared with the state-of-art algorithms. The heart disease data sets are taken as input for our PSNB algorithm collected from the UCI machine learning repository [38]. The average accuracy of *PSNB* algorithm for both batch and streaming data are shown in Fig. 5. Initially, the average accuracy is low for all the comparative algorithms when the number of patients are less than 100. As the number of patient increases, the average prediction accuracy is increased gradually and has a convergence trend of all these algorithms. The average accuracy of *PSNB* prediction is 12.8% higher than that of original *NB* algorithm and is 23.8% higher than *ANN* algorithm when the number of patients is equal to 300. Therefore, it is observed that the PSNB algorithm accuracy is improved significantly and the trend continues for the rest of the patients.

ACCEPTED MANUSCRIPT

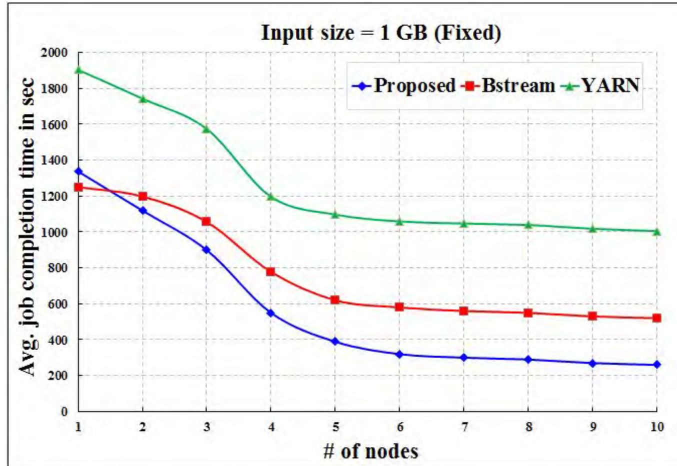

Figure 6: Average processing time for different cluster size.

In our observation, processing or job completion time is an important parameter to observe the efficiency of the algorithm as shown in Fig. 6. In our proposed model, the job completion time is defined as the sum of scheduling, execution and transfer time of the jobs on different nodes within ID and ED. It is observed that the average job completion time of our proposed model is less than Hadoop YARN [39] and Bstream [18], respectively when the number of nodes are high in ID and ED. For small number of nodes, the processing time of the proposed model is just above the Btream, though it is less than the Hadoop YARN. **However, increase in the number of nodes does not enhance the processing time due to sparse data and as it needs more time for execution. The saturation point for job completion time is achieved after 7 number of nodes.** Therefore, the processing time is directly proportional to the scheduling, execution and data transfer time.

The average execution time of the job is shown in Fig. 7. **For small data size, i.e., less than 0.40GB ($\cong$ 400 MB), the execution times of Bstream is less than the proposed model as a fixed amount of time is required for the Spark cluster's setup and configuration.** Initially, the Batch and Block intervals are decided for the incoming data sets, which take some time to find the optimum case. **However, a noticeable execution time gap is observed for the larger data size $>$ 0.40GB, and the trend continues until it reaches up to 1 GB.** Hence, our proposed model has lower execution time than Bstream and Hadoop YARN

for large data sets as most of the jobs achieve data locality in IDs and EDs. Moreover, the Spark based healthcare big data processing platform has significant strength over Bstream and YARN due to in-memory and parallel execution of the jobs.

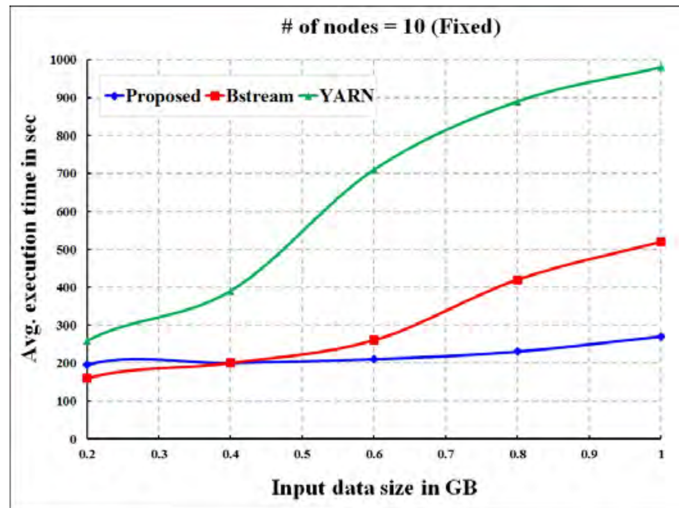

Figure 7: Average execution time for different data size.

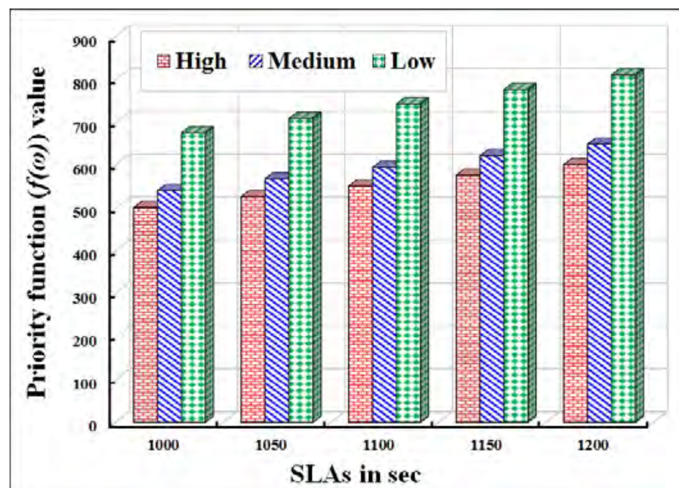

Figure 8: Priority function evaluation for different SLAs.

The priority of the jobs are decided by the priority weight function $(f(w))$

as shown in Fig. 8, where SLA, current health condition, disease severity and emergency factor of a patient are taken as input for the evaluation. It is clearly observed that the value of $f(w)$ is increased with the increase in SLA. The $f(w)$ value is found to be 501 for 1000 as value of SLA, where α, ψ and ϵ values are 0.2, 0.2 and 0.1, respectively, which has the highest priority. Similarly, for same SLA value, with different α, ψ and ϵ values such as 0.9, 0.9, and 0.9, $f(w)$ is calculated as 674, which has the lowest priority. The medium priority is evaluated as 540 when value of α, ψ and ϵ is 0.2, 0.8, and 0.9, respectively. For lower values of α, ψ and ϵ the condition of the patient is critical. Hence, the lower value of $f(w)$ is considered as the highest priority.

In Fig. 9, the violation of SLA for different priority jobs in ID and ED is displayed. The numbers of violations of low priority jobs are higher than the high priority jobs as the high priority jobs are executed earlier in order. Even the number of violations are further reduced by incorporating the EDs during heavy load. For instance, 6 and 3 numbers of low priority jobs are failed to satisfy the SLA during execution of 50 jobs in only ID and ID with ED, respectively. However, less numbers of high priority jobs are failed such as 2 and 1 out of total 50 number of jobs, executed in only ID and ID with EDs, respectively. Hence, the throughput of the system is increased by considering both ID and ED in the cloud.

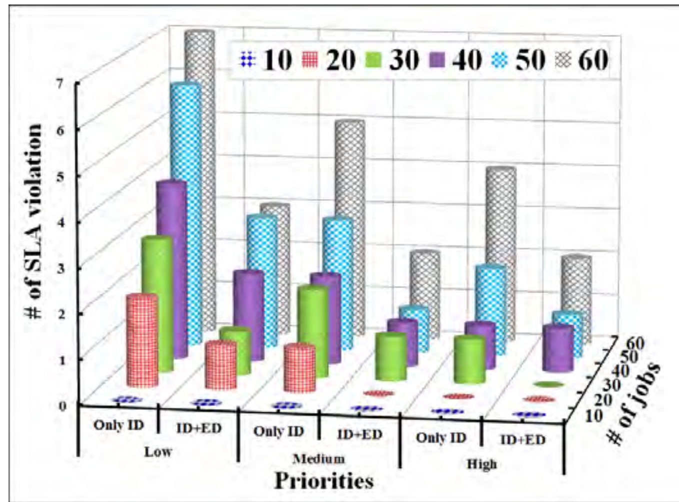

Figure 9: Violation of SLA for different priority jobs in ID and ED.

In Fig. 10, the speedup of the proposed model is viewed for different clus-

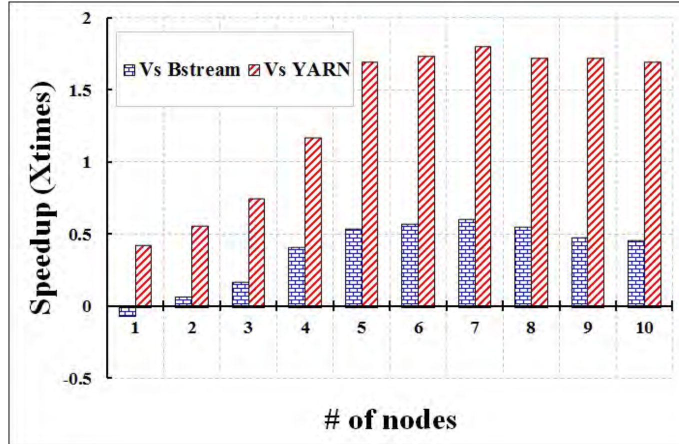

Figure 10: Speedup comparison of Proposed model with BStream and YARN.

ter size and is compared with the existing data processing models. The speedup is calculated by taking the ratio of execution time of the standalone and cluster mode. By considering the in-memory, parallel execution in cloud-based Spark platform, the speedup of our proposed model tends to increase with the increase in the number of worker nodes. **For instance, the speedup of our proposed model almost touches to 0.6 and 1.8 as compared to Storm and YARN, respectively, when the number of worker nodes is equal to 7. Thereafter, there is a saturation on the speedup even though the number of node is increased to 10.** Hence, our proposed model can process the patient data speedily within the *SLA*.

As shown in Fig. 11, the communication cost of the proposed model is evaluated with respect to Spark-MLRF [23]. It is observed that the shuffle write of the proposed model is less than that of Spark-MLRF and it outperforms for more number of nodes. For instance, if the number of worker nodes is increased from 1 to 10, the shuffle write of Spark-MLRF is increased from 75MB to 420MB. However, the shuffle write of the proposed model is increased slowly from 10.5MB to 80MB, which becomes steady with the increase in the number of worker nodes. This is due to the data locality achieved by the ODD mechanism. There is a significant reduction of data communication overhead and therefore, our proposed model outperforms as compared to other parallel pro-

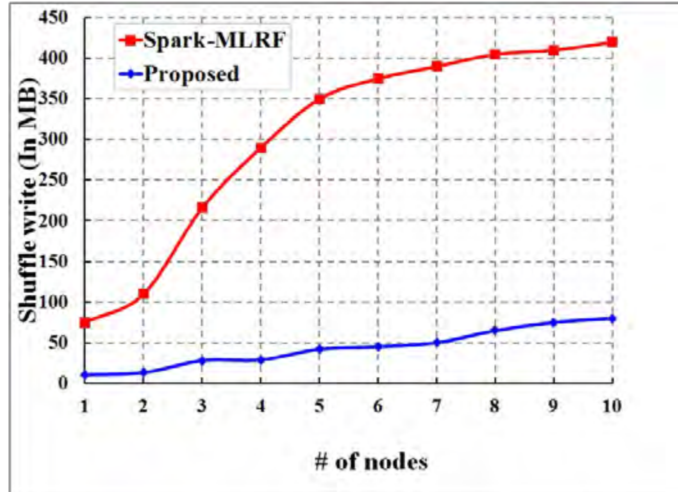

Figure 11: Communication using shuffle write.

cessing models.

7. Conclusion

In this work, *SLA* based healthcare big data analysis and computing model is proposed, where both batch and streaming patient data are analyzed in *ID* and *ED* environment. A *PSNB* method is proposed for healthcare Big Data analysis and a *MCA* algorithm is also designed for dimension reduction to improve the accuracy of *PSNB* algorithm. Besides, a *GPS* algorithm is proposed to prioritize the emergency jobs of the patients. To achieve data locality, an *ODD* algorithm is proposed in this paper. To improve the execution time of the prioritized jobs, a *DJS* algorithm is proposed that satisfies the *SLA*. Implementation results show that our forefront healthcare big data analytic model and proposed algorithms have outperformed in terms of accuracy and overall job completion time. The proposed model can be conveniently employed in various medical applications specifically for emergency patient data processing and analysis.

Acknowledgment

This work is partly supported by Ministry of Science and Technology (MOST), Taiwan under the grant number 106-2221-E-182-014, 105-2221-E-182-050 and 105-2221-E-182-043.

ACCEPTED MANUSCRIPT

References

[1] N. Kaur, S. K. Sood, An energy-efficient architecture for the internet of things (IoT), IEEE Systems Journal PP (99) (2015) 1–10.

[2] S. K. Mohapatra, P. K. Sahoo, S.-L. Wu, Big data analytic architecture for intruder detection in heterogeneous wireless sensor networks, Journal of Network and Computer Applications 66 (2016) 236 – 249.

[3] M. D. Assuno, R. N. Calheiros, S. Bianchi, M. A. Netto, R. Buyya, Big data computing and clouds: Trends and future directions, Journal of Parallel and Distributed Computing 79-80 (Supplement C) (2015) 3 – 15, special Issue on Scalable Systems for Big Data Management and Analytics.

[4] M. M. Al-Sayed, S. Khattab, F. A. Omara, Prediction mechanisms for monitoring state of cloud resources using markov chain model, Journal of Parallel and Distributed Computing 96 (Supplement C) (2016) 163 – 171.

[5] D. Singh, D. Roy, C. K. Mohan, Dip-svm : Distribution preserving kernel support vector machine for big data, IEEE Transactions on Big Data 3 (1) (2017) 79–90.

[6] N. Bharill, A. Tiwari, A. Malviya, Fuzzy based scalable clustering algorithms for handling big data using apache spark, IEEE Transactions on Big Data 2 (4) (2016) 339–352.

[7] P. K. Sahoo, S. K. Mohapatra, S. L. Wu, Analyzing healthcare big data with prediction for future health condition, IEEE Access 4 (2017) 9786–9799.

[8] S. Ramirez-Gallego, S. Garcia, H. Mourino-Talin, D. Martinez-Rego, V. Bolon-Canedo, A. Alonso-Betanzos, J. M. Benitez, F. Herrera, Distributed entropy minimization discretizer for big data analysis under apache spark, in: 2015 IEEE Trustcom/BigDataSE/ISPA, Vol. 2, 2015, pp. 33–40.

[9] B. Tang, H. He, P. M. Baggenstoss, S. Kay, A bayesian classification approach using class-specific features for text categorization, IEEE Transactions on Knowledge and Data Engineering 28 (6) (2016) 1602–1606.

ACCEPTED MANUSCRIPT

[10] F. Zheng, G. Webb, A comparative study of semi-naive bayes methods in classification learning, in: Proceedings of the Fourth Australasian Data Mining Conference (AusDM05), University of Technology, Sydney, 2005, pp. 141–156.

[11] A. Amokrane, R. Langar, M. F. Zhani, R. Boutaba, G. Pujolle, Greenslater: On satisfying green slas in distributed clouds, IEEE Transactions on Network and Service Management 12 (3) (2015) 363–376.

[12] H. Alshammari, J. Lee, H. Bajwa, H2hadoop: Improving hadoop performance using the metadata of related jobs, IEEE Transactions on Cloud Computing (99).

[13] T. Li, J. Tang, J. Xu, Performance modeling and predictive scheduling for distributed stream data processing, IEEE Transactions on Big Data 2 (4) (2016) 353–364.

[14] M. Zaharia, An Architecture for Fast and General Data Processing on Large Clusters, Association for Computing Machinery and Morgan; Claypool, New York, NY, USA, 2016.

[15] F. Zhang, J. Cao, S. U. Khan, K. Li, K. Hwang, A task-level adaptive mapreduce framework for real-time streaming data in healthcare applications, Future Generation Computer Systems 43-44 (2015) 149 – 160.

[16] K. Hildebrandt, F. Panse, N. Wilcke, N. Ritter, Large-scale data pollution with apache spark, IEEE Transactions on Big Data PP (99) (2017) 1–1.

[17] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauly, M. J. Franklin, S. Shenker, I. Stoica, Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing, in: Presented as part of the 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12), USENIX, San Jose, CA, 2012, pp. 15–28.

[18] S. Kailasam, P. Dhawalia, S. J. Balaji, G. Iyer, J. Dharanipragada, Extending mapreduce across clouds with bstream, IEEE Transactions on Cloud Computing 2 (3) (2014) 362–376.

ACCEPTED MANUSCRIPT

[19] S. Kang, B. Veeravalli, K. M. M. Aung, Dynamic scheduling strategy with efficient node availability prediction for handling divisible loads in multi-cloud systems, Journal of Parallel and Distributed Computing 113 (2018) 1 – 16.

[20] H. Chen, F. Z. Wang, Spark on entropy: A reliable efficient scheduler for low-latency parallel jobs in heterogeneous cloud, in: 2015 IEEE 40th Local Computer Networks Conference Workshops (LCN Workshops), 2015, pp. 708–713.

[21] J. Andreu-Perez, C. C. Y. Poon, R. D. Merrifield, S. T. C. Wong, G. Z. Yang, Big data for health, IEEE Journal of Biomedical and Health Informatics 19 (4) (2015) 1193–1208.

[22] M. Xu, H. Chen, P. K. Varshney, Dimensionality reduction for registration of high-dimensional data sets, IEEE Transactions on Image Processing 22 (8) (2013) 3041–3049.

[23] J. Chen, K. Li, Z. Tang, K. Bilal, S. Yu, C. Weng, K. Li, A parallel random forest algorithm for big data in a spark cloud computing environment, IEEE Transactions on Parallel and Distributed Systems 28 (4) (2017) 919–933.

[24] J. Archenaa, E. A. M. Anita, Interactive Big Data Management in Healthcare Using Spark, Springer International Publishing, Cham, 2016.

[25] Y.-C. Kao, Y.-S. Chen, Data-locality-aware mapreduce real-time scheduling framework, J. Syst. Softw. 112 (C) (2016) 65–77.

[26] Z. Tang, X. Zhang, K. Li, K. Li, An intermediate data placement algorithm for load balancing in spark computing environment, Future Generation Computer Systems.

[27] K. Wang, Z. Bian, Q. Chen, Millipedes: Distributed and set-based subtask scheduler of computing engines running on yarn cluster, in: High Performance Computing and Communications (HPCC), IEEE 17th International Conference on, 2015, pp. 1597–1602.

[28] M. Zaharia, D. Borthakur, J. Sen Sarma, K. Elmeleegy, S. Shenker, I. Stoica, Delay scheduling: A simple technique for achieving locality and fairness in cluster scheduling, in: Proceedings of the 5th European

ACCEPTED MANUSCRIPT

Conference on Computer Systems, EuroSys '10, ACM, New York, NY, USA, 2010, pp. 265–278.

[29] M. Zaharia, A. Konwinski, A. D. Joseph, R. Katz, I. Stoica, Improving mapreduce performance in heterogeneous environments, in: Proceedings of the 8th USENIX Conference on Operating Systems Design and Implementation, OSDI'08, USENIX Association, Berkeley, CA, USA, 2008, pp. 29–42.

[30] N. Zaheilas, V. Kalogeraki, Real-time scheduling of skewed mapreduce jobs in heterogeneous environments, in: 11th International Conference on Autonomic Computing (ICAC 14), USENIX Association, Philadelphia, PA, 2014, pp. 189–200.

[31] A. Reuther, C. Byun, W. Arcand, D. Bestor, B. Bergeron, M. Hubbell, M. Jones, P. Michaleas, A. Prout, A. Rosa, J. Kepner, Scalable system scheduling for HPC and big data, Journal of Parallel and Distributed Computing 111 (2018) 76 – 92.

[32] B. Lin, W. Guo, N. Xiong, G. Chen, A. V. Vasilakos, H. Zhang, A pretreatment workflow scheduling approach for big data applications in multicloud environments, IEEE Transactions on Network and Service Management 13 (3) (2016) 581–594.

[33] C.-H. Weng, T. C.-K. Huang, R.-P. Han, Disease prediction with different types of neural network classifiers, Telematics and Informatics 33 (2) (2016) 277 – 292.

[34] N. Bayasi, T. Tekeste, H. Saleh, B. Mohammad, A. Khandoker, M. Ismail, Low-power ecg-based processor for predicting ventricular arrhythmia, IEEE Transactions on Very Large Scale Integration (VLSI) Systems 24 (5) (2016) 1962–1974.

[35] S. Francesco, B. Alberto, C. F. C.M., F. Valeria, T. Flavio, R. Riccardo, R. Gianluca, C. Claudio, M. Claudio, D. Domenico, P. Roberto, Tumor size as a prognostic factor in patients with stage IIa colon cancer, The American Journal of Surgery 215 (1) (2018) 71 – 77.

[36] Spark streaming programming guide (2017).
URL `https://spark.apache.org/docs/latest/streaming-programming-guide.html`

ACCEPTED MANUSCRIPT

[37] M. Deshpande, G. Karypis, Using conjunction of attribute values for classification, in: Proceedings of the Eleventh International Conference on Information and Knowledge Management, CIKM '02, ACM, New York, NY, USA, 2002, pp. 356–364.

[38] M. Lichman, UCI machine learning repository (2013).
URL http://archive.ics.uci.edu/ml

[39] C. Xu, R. Goldstone, Z. Liu, H. Chen, B. Neitzel, W. Yu, Exploiting analytics shipping with virtualized mapreduce on hpc backend storage servers, IEEE Transactions on Parallel and Distributed Systems 27 (1) (2016) 185–196.

ACCEPTED MANUSCRIPT

*Author Biography & Photograph

Suvendu Kumar Mohapatra received B.Tech degree from Biju Pattnaik University, India, in June 2008, the M.Tech degree from IIIT, Bhubaneswar, in June 2010 and PhD degree in the department of Electrical Engineering, Division of Computer Science and Information Engineering, Chang Gung University, Taiwan. He is currently an Assistant Professor in Industry 4.0 implementation center of National Taiwan University of Science and Technology, Taiwan. His research interests include the areas of Big Data Analysis with cloud: Medical big data analysis, Prediction, Optimization and Machine Learning.

Prasan Kumar Sahoo received Master of Science in Mathematics from Utkal University, India in 1994 and Master of Technology in Computer Science from Indian Institute of Technology (IIT), Kharagpur, India in 2000. He received the first PhD in Mathematics from Utkal University, India, and second PhD in Computer Science and Information Engineering from National Central University, Taiwan in 2002 and 2009, respectively. He is currently a Full Professor in the department of Computer Science and Information Engineering and Director of International Cooperation Center of Chang Gung University, Taiwan. He was Associate Professor in the department of Information Management, Vanung University, Taiwan and has worked in the Software Research Center of National Central University, Taiwan. His current research interests include Big Data analytic, Cloud Computing and Cyber-physical Systems. He is an Editorial Board Member of International Journal of Vehicle Information and Communication Systems (IJVIC) and has served as the Program Committee Member of several IEEE and ACM conferences. He is Senior Member, IEEE and was Program Chair of ICCT, 2010.

ACCEPTED MANUSCRIPT

Shih-Lin Wu received the B.S. degree in Computer Science from Tamkang University, Taiwan, in June 1987 and the Ph.D. degree in Computer Science and Information Engineering from National Central University, Taiwan, in May 2001. Since 2000, He joined the Department of Computer Science and Information Engineering, Chang Gung University. He is Full Professor (2015 preset) and Chairman (2016 present) at the Department of Computer Science and Information Engineering, Chang Gung University. His current research interests include mobile communications, wireless networks, wireless ad hoc networks, and intelligent robots. He serves as a member of editor board of Telecommunication Systems, Journal of Positioning and ISRN Communications. He was a Guest Editor of International Journal of Pervasive Computing and Communications 2007, a Program Chair of Mobile Computing 2005, a Program Chair of International Workshop on Data Management in Ad Hoc and Pervasive Computing 2009, a Co-Chair of International High Speed Intelligent Communication 2009, and a Co-Chair of International Symposium on Bioengineering 2011, a General Chair of Mobile Computing 2012, a Co-Chair of International High Speed Intelligent Communication and International Conference on Computational Problem-Solving 2013, a Special Session Chair of International Conference on Advanced Robotics and Intelligent Systems 2014, a Special Session Chair of International Conference on Telecommunication Systems Management 2014, and a Program Chair of International Computer Symposium 2016,. Several of his papers have been chosen as Selected/Distinguished papers in international conferences. Dr. Wu is a member of the IEEE and the Phi Tau Phi Society.