View metadata, citation and similar papers at core.ac.uk
brought to you by CORE
provided by SEDICI - Repositorio de la UNLP

Automatic and early detection of the deterioration of patients in Intensive and Intermediate Care Units: technological challenges and solutions

Javier Balladini[1], Pablo Bruno[1], Rafael Zurita[1], and Cristina Orlandi[2]

[1] *Universidad Nacional del Comahue, Neuquén, Argentina*
{javier.balladini, rafa}@fi.uncoma.edu.ar, pablo.bruno@est.fi.uncoma.edu.ar
[2] *Hospital Francisco Lopez Lima, Río Negro, Argentina*
orlandi.mariacristina@gmail.com

Abstract

In the Intensive and Intermediate Care Units of healthcare centres, many sensors are connected to patients to measure high frequency physiological data. In order to analyse the state of a patient, the medical staff requires both appropriately presented and easily accessed information. As most medical devices do not support the extraction of digital data in known formats, medical staff need to fill out forms manually. The traditional methodology is prone to human errors due to the large volume of information, with variable origins and complexity. The automatic and real-time detection of changes in parameters, based on known medical rules, will make possible to avoid these errors and, in addition, to detect deterioration early. In this article, we propose and discuss a high-level system architecture, an embedded system that extracts the electrocardiogram signal from an analog output of a medical monitor, and a real-time Big Data infrastructure that integrate Free Software products. We believe that the experimental results, obtained with a simple prototype of the system, demonstrate the viability of the techniques and technologies used, leaving solid foundations for the construction of a reliable system for medical use, able to scale and support an increasing number of patients and captured data.

Keywords: Intensive Care Unit, Clinical Decision Support System, Medical Rules Processing, Big Data, Embedded System.

Resumen

En las unidades de cuidados intensivos e intermedios de centros de salud, muchos sensores están conectados a los pacientes para medir datos fisiológicos de alta frecuencia. Para analizar el estado de un paciente, el personal médico requiere información presentada de manera apropiada y de fácil acceso. Como la mayoría del equipamiento médico no admite la extracción de datos digitales en formatos conocidos, el personal médico completa formularios manualmente. Esta metodología es propensa a errores humanos debido al gran volumen de información, con orígenes y complejidad variable. La detección automática y en tiempo real de cambios en los parámetros, basados en reglas médicas conocidas, permitirá evitar estos errores y, además, detectar el deterioro de forma temprana. En este artículo, proponemos una arquitectura de alto nivel del sistema, un sistema embebido que extrae la señal del electrocardiograma de una salida analógica de un monitor médico, y una infraestructura Big Data de tiempo real que integra productos Software Libre. Creemos que los resultados experimentales, obtenidos con un prototipo, demuestran la viabilidad de las técnicas y tecnologías utilizadas, dejando sólidas bases para la construcción de un sistema confiable para uso médico, y capaz de escalar para soportar un número creciente de pacientes y datos capturados.

Palabras claves: Unidad de Cuidados Intensivos, Sistema de soporte a la decición clínica, Procesamiento de reglas médicas, Big Data, Sistema embebido.

1 Introduction

In health centres, the Intensive Care Unit (ICU) provides comprehensive, rigorous, and continuous care for adult persons who are critically ill and who can benefit from treatment, providing a good die for unrecoverable patients. Some health institutions, in turn, have an Intermediate Care Unit (IMCU) that, unlike the ICU, provides care to patients who do not require life-sustaining therapeutic treatments, but to monitoring and control.

In both units, all patients are connected to a monitor that measures their vital signs and issues alerts that indicate a risk to their health. These alerts are determined under criteria based on the standard population. In the case of the ICU, patients could additionally be connected to other medical equipment. Some of them, such as mechanical respirators, also issue risk alerts. Almost all alerts issued by a medical equipment are

only based on parameters that it measures. Exceptionally, one device can be interconnected with another to issue alerts based on parameters measured by both devices.

In order to analyse the state of a patient, the medical staff require both appropriately presented and easily accessed information. As most medical devices do not support the extraction of digital data in known formats, medical staff need to fill out forms manually. These forms are used to record data observed in each equipment at hourly intervals, and to record data related to medical studies (as laboratory tests). At the end of each day, the physician analyse the data of the forms and produce indications for the nurses, such as modifications in the medication, practices to be performed on patients, etc.

Our objective is to develop a computer system that allows the automatic and early detection of deterioration of patients hospitalised in ICUs and IMCUs, through the real-time processing and analysis of digital health data. The data are acquired from different sources, including the real-time data extraction from medical equipment.

The traditional methodology is prone to human errors due to the large volume of information, with variable origins and complexity, that the medical staff must analyse. The automatic and real-time detection of changes in parameters (from multiple digital sources: software systems and medical equipment), based on known medical rules, will make it possible to avoid these errors and, in addition, to detect deterioration early. The latter will give physicians the ability to plan and begin treatments without delays, possibly increasing their effectiveness (and consequently reducing mortality) and decreasing their economic cost. Additionally, the accuracy of the diagnoses could be increased by contemplating the totality of the data generated by the medical equipment connected to the system, instead of a few manually registered. In the long term, the historical record of data will be extremely valuable for conducting studies to discover patterns in the data that can predict pathologies, such as septic shock [1].

In the literature, few systems of this type are found, some of them are [2, 3, 4, 5, 6, 7]. However, its authors have not exposed, to the community, accurate information on how they have addressed the different challenges inherent to the system and/or the performance achieved by their solutions, or they use proprietary software or hardware (for data acquisition), or they do not meet the requirements of: scalability, fault tolerance, and interoperability.

In this paper we discuss challenges related to the construction of the system and propose solutions that were implemented in our prototype. It is designed to be deployed at the Francisco Lopez Lima Hospital, a public hospital with relatively high financial constraints, located in the city of General Roca, Río Negro, Argentina.

The main contributions of this work are focused on the proposal of:

1. A high-level system architecture, which support multiple hospitals with integrated data storage and knowledge extraction, considering hospitals without Internet leased lines. It uses a mix of local computing and a public cloud (to reduce economic cost). An initial approach was proposed in [8].

2. An embedded device that extracts the electrocardiogram (ECG) signal from an analog output of a medical monitor, performs an analog-digital conversion, and transmits it via WiFi to the platform that process the signal. This device seeks to be an alternative to the problem of digital data extraction of medical equipment produced by the use of different communication interfaces, mostly proprietary, and whose specifications are not published by the manufacturers.

3. A real-time Big Data infrastructure that, based on streams of signals data and other health data, allows to process rules to determine and issue alerts indicating risk in the health of patients. A distributed, scalable, fault tolerant and interoperable solution is proposed, based on the integration of software products under Free Software licenses. It is included a web system that allows user interfaces with visualisation of signals and alerts in real-time.

The rest of this paper is organised as follows. Section 2 describes the high-level architecture of the system. Section 3 presents the embedded system that acquire the ECG signal from a medical monitor. Section 4 discusses the Big Data infrastructure that support real-time processing, analysis, and visualisation of health data. Finally, section 6 presents the conclusions and future works.

2 High-level System Architecture

Figure 1 shows the high-level system architecture proposed. In this figure it is observed several Hospitals connected to a Public Cloud. As we are considering hospitals with unreliable Internet access, the critical part of the system requires to perform on a dedicated computing system placed inside each hospital. In a hospital, the component Data Acquisition is in charge of the connection between our system and external health data sources: Medical Equipment and the Electronic Health Record (EHR) System. The Data Acquisition module streams the data to a processing infrastructure. The infrastructure must be capable of processing, in real time, a large volume of data per unit of time. For that reason it receives the name of

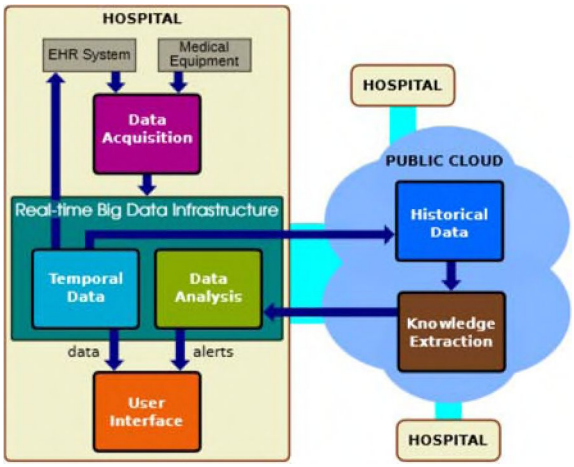

Figure 1: High-level System Architecture

"Real-time Big Data Infrastructure". This large volume of data per unit of time comes mainly from signals produced by medical equipment. For example, one of these signals is the electrocardiogram (ECG), a test that records the electrical activity of the heart over a period of time using electrodes placed on the pacient skin, and is used to detect cardiac problems. A typical vital signs monitor takes 1,000 samples per second, and advanced converters can sample at 10,000 to 15,000 per second or even higher [9].

The Real-time Big Data Infrastructure must manage Temporal Data (see Figure 1), which are health data relevant to the system, corresponding to patients currently hospitalised. All data acquired from the time of the patient's admission to the ICU/IMCU are considered. When the patient leaves the inpatient unit, some data may be recorded in the EHR system.

Some data are useful by themselves, and other data may need to be analysed to produce new data. For example, the signals obtained from medical equipment could be interpreted through a process of Data Analysis (see figure 1), generating meta-data, that is, new health data of patients. This is the case of parameters such as heart rate, which comes from the analysis of the ECG. The system also performs another type of data analysis: the processing of medical rules. In the figure 2 two example rules are shown. Each rule defines the conditions, relating parameters and values, which must be met to generate an alert indicating risk in the patient's health. Each patient can be associated with a particular set of rules. Alerts are classified according to a risk coefficient that determines their importance; the lower the value, the greater the risk to patient health and, therefore, the greater the importance of the alert. Each alert has a description, possibly different, for physicians and nurses. One of both alert descriptions could be null, allowing to support alerts directed to a single group, physicians or nurses. Optionally, each of these groups is associated with a treatment plan to be carried out. The rule also determines the activation or deactivation of other rules (including itself), possibly after a certain time. Once an episode of deteriorating patient health is overcome, certain rules are re-activated.

The User Interface allows nurses and physician to receive alerts, and to visualise any Temporal Data such as raw data (signals, vital signs, etc.) or statistical processed data (tables, charts, etc.). It is possible to view current raw data (in real-time) and to explore previous data.

Our system includes the storage of Historical Data for Knowledge Extraction, useful for future medical research and the discovery of patterns for predicting pathologies. The Knowledge Extraction is directed by physicians, and the output of this component are clinical rules and data specifically tailored to enable physicians to perform clinical research. The new rules are then incorporated to the set of rules used for Data Analysis at each hospital.

The Historical Data includes all acquired inpatient data from any hospital. This data is saved in a persistent way in a Public Cloud. It must be observed that the temporal storage on hospitals needs to save only data of current inpatients. This means the storage does not need to scale simply by the passing time, but only when grows the amount of beds or the number of considered health parameters (possibly with new acquired data from medical equipment). Instead, the persisting storage requires to scaling with the passing time because it must save the new medical data that are received from hospitals. Therefore, the scalable storage offered by a public cloud is very useful. The computing resources of the public cloud can be used to perform the Knowledge Extraction. As intensive computation for Knowledge Extraction need to be done sporadically, a Public Cloud will be effective in cost (avoiding the high costs of a dedicated system and technicians).

3 Data acquisition

The acquisition of data from an EHR system does not present significant technological problems because they are normally prepared for interoperability. The challenge is in the data acquisition from medical equipment, the lowest level component (hardware and software) of the whole system presented on this work. It must collect signals from medical monitors (INPUT), and send the digital information (OUTPUT) to the Real-time Big Data Infrastructure, to be processed. If analog signals exist a conversion to a proper digital representation is required before the transfer.

Interfacing with the complete set of equipments is the long term goal for this level. Unfortunately, the hospital keep using a wide and complex variete of ancient and newer monitoring equipments, so different INPUT/OUTPUT interfaces exist. In some cases, two models from a same manufacturer (but different year of

```
RULE 1
IF (RR > 30 per minute)
   and (Hyperlactacidemia:  > 3 mmol/L)
   and (Arterial Hypotension:  SBP < 90 mmHg or MAP < 70 mmHg)
   and (Fever > 38 ℃)
THEN
   Risk coefficient:  2
   Nurse alert:  Probable Septic Shock
   Nurse treatment:  Infuse fluid 20 ml/Kg
   Physician alert:  Probable Septic Shock
   Physician treatment:  Two blood cultures and start antibiotic therapy
   Activate rules:  2 after 30 minutes
   Deactivate rules:  1

RULE 2
IF (Arterial Hypotension:  SBP < 90 mmHg or MAP < 70 mmHg)
THEN
   Risk coefficient:  1
   Nurse alert:  Septic Shock
   Nurse treatment:  Start noradrenaline infusion:  16mg/250cc dextrose 5% 21ml/Hour ⇒ Objective MAP 70 mmHg
   Physician alert:  Septic Shock
   Physician treatment:  no
   Activate Rules:  3
   Deactivate Rules:  2

References:
   RR = Respiratory Rate
   SBP = Systolic Blood Pressure
   MAP = Mean Arterial Pressure
```

Figure 2: Examples of medical rules

production) do not use identical hardware ports and/or software protocols for the output signals. Thus the data extraction from medical equipment raises a research topic previous to the design of an embedded system for data acquisition, because internal specifications for those variety of interfaces are not always published by the manufacturers. Either since it uses propietary protocols, or the proper documentation might not available (if, e.g. an ancient equipment is not supported anymore). As a consequence, the data acquisition might not be completely possible for all the equipments.

Many monitors uses the RS-232 standard, for the electrical and mechanical characteristics of outputs. Differences here ocurres on the internal data level. RS-232 is commonly used for serial communication betweens systems, but some equipment might work internally with other different protocols, for interconnecting devices from the same vendor only.

In order to develop and test our first data acquisition embedded system prototype we choose, for interfacing, the Life Scope LC, BSM-3101, from Nihon Kohden. The BSM-3101 is a medical monitor, with an analog interface for continuos ECG data output. Since it features a non-digital output (which requires to be converted) this equipment is suitable for testing the longest use case (list of actions) of our data acquisition prototype. Interfacing other digital outputs monitors might be straighforward (if the correct documentation is available).

3.1 Embedded ECG Signal Acquisition System

The data acquisition system prototype architecture is shown on figure 3. It comprises a microcontroller and a single board computer (SBC). The former is a 8-bit

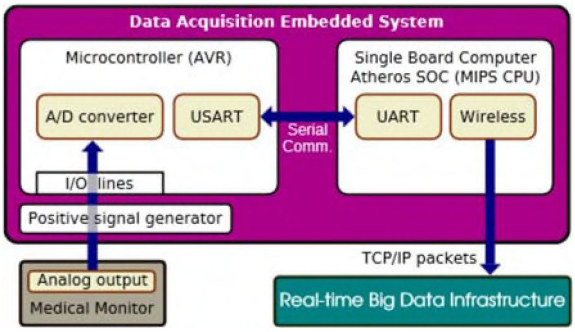

Figure 3: Data Acquisition Embedded System Architecture

CPU (AVR architecture) with several low level I/O lines, set for interfacing with monitors. It also features 2KB SRAM, 32KB flash memory, a 6-channel 10-bit analog-to-digital converter, SPI serial port, a two-wire serial interface, and a serial programmable USART. The selected SBC includes a Wi-SOC from Atheros (32-bit 400Mhz MIPS CPU, 32MB RAM, 4MB flash memory) with low power consumption and reliable Wi-Fi interface. The Wi-SOC is the wireless communication bridge between the whole data acquisition device and a central server.

The two components communicate using the Universal Asynchronous Receiver/Transmitter (UART). The maximum bits/baud rate per second is 115,200, which represents almost 100kbits per second. In case of there is an excessive continous data input, the UART would be the limiting hardware on this architecture. However, it is planified to use just one of this low cost data acquisition device per patient/bed, so there should not be greater input data than the limit imposed by the UART.

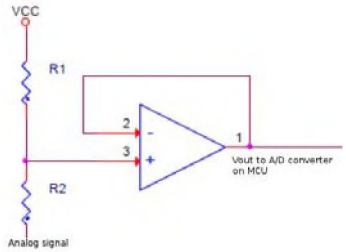

Figure 4: Positive signal generator

On the software level, a custom firmware for the microcontroller was developed. It reads the analog signal from ECG using an analog-to-digital converter driver. Then, after the conversion, the MCU transfers the 10-bit resolution digital value to the SBC using an UART driver. Since there is not other software tasks so far, the INPUT action is accomplished using polling programming, inside an infinite loop. When some input data is available, the infinite loop calls the send procedure, which is part of the UART driver.

It was observed (when studying the ECG monitor internals) that the analog output has positive and negative voltages (-5v to 5v range). In view of the fact that the choosen MCU is not able to read the negative voltage (the ADC works on 0v to 5v range) the analog signal was mounted on a unipolar positive signal generator little circuit, which is achieved using a voltage divisor and a operational amplifier (OpAmp). Figure 4 shows this circuit.

The software in the SBC includes two main components: a custom Linux kernel (featuring UART and wireless drivers, and TCP/IP software layers), and the userspace software. The latter was built using buildroot project, which is suitable for small Linux devices with low memory. On userspace there is also a custom application, which uses the Linux UART driver to read for incoming digital data from MCU. When digital data bytes are read the userspace application transfers those to the central server, using a TCP/IP connection.

3.2 ECG Signal Acquisition System Validation

Several measurement tests of rate and precision were performed in a real environment at the hospital. It is known that thousand samples per second from the ECG are an adequate amount for describing the patient condition on real-time. This rate of data was taken for several hours on this actual environment with no lost of information, and a wireless TCP/IP online communication during the whole test.

The most important validation is about the precision of data acquired. For this purpose a comparison between results obtained by the prototype and a real oscilloscope (PicoScope 2203) was made. Many samples for several seconds were taken using both data acquisition systems, at the same moment. All the samples were saved, and a script was used to graphically show the representative curves (useful for medical diagnosis, and for our validation). When both curves were overlapped it was demostrated that the prototype is acquiring the data from ECG correctly. Figure 5 shows the ECG over 2 sec, with the two set of samples graphically overlapping. The green curve (on top of the violet curve) are the samples taken by our prototype. The violet curve the samples gotten by the PicoScope.

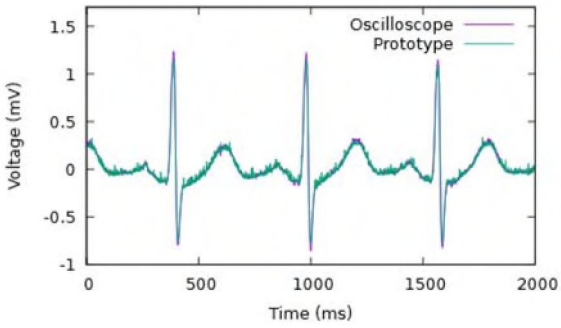

Figure 5: ECG prototype validation

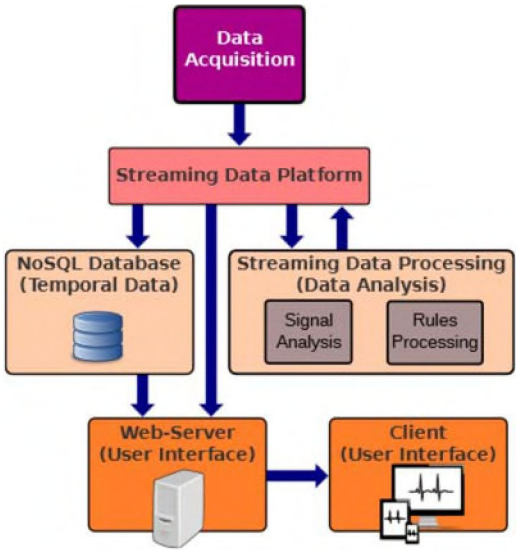

Figure 6: Architecture of the Real-time Big Data Infrastructure

4 Real-time Big Data Infrastructure

The Real-time Big Data Infrastructure, whose functionality was described in section 1, is implemented using the architecture shown in figure 6. Data are organised in a central platform, the Streaming Data Platform, which receives data streams and makes them available to other components to be consumed in real time. It works as a messaging system or message queue, under the publication-subscription pattern. This organisation of the data allows to simplify the flow of communications between the different components, producing a

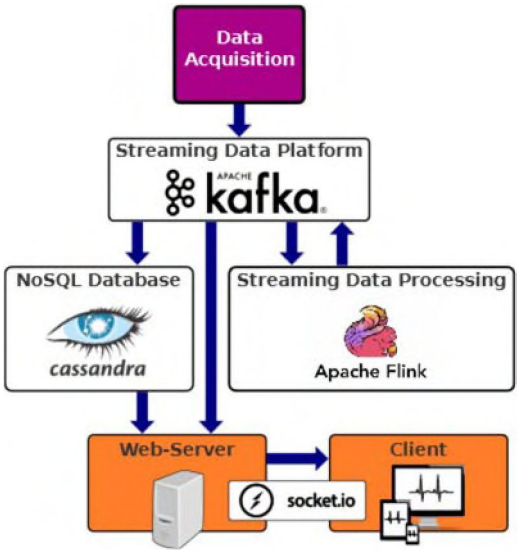

Figure 7: Technologies of the Real-time Big Data Infrastructure

low coupling between them.

The Data Acquisition module extracts data from the EHR System and the Medical Equipment, and sends them in the form of streams to the central platform. Most of data received by the platform comes from the Data Acquisition.

Streaming Data Processing consumes data from the central platform, and is responsible for performing a Data Analysis, that is, the analysis of physiological signals and the processing of medical rules. The results of the processing/analysis are returned to the central platform.

The NoSQL Database allows the storage of Temporary Data. This type of database (NoSQL) are designed to store and process big data, with high-performance reading and writing operations [10]. The NoSQL Database consumes data from the central platform (raw data produced by the Data Acquisition and data generated by Streaming Data Processing) and writes them to secondary storage.

Finally, the User Interface module presents the data (signals, vital signs, etc.) to the physicians and nurses, possibly making a small prior processing of them when statistical data are required. This module can receive data from the Streaming Data Platform or NoSQL Database, depending on whether the required data are real time or past time, respectively.

The figure 7 shows the software products, with Free Software licenses, selected for the implementation of the Real-time Big Data Infrastructure. The following sections describe the operation of each used software, and how they are integrated into our prototype.

4.1 Streaming Data Patform

The Streaming Data Platform is implemented by Apache Kafka, a distributed streaming platform that

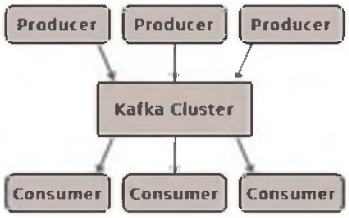

Figure 8: Kafka Producers and Consumers

handles data streams in real time [11]. Kafka was originally developed at LinkedIn and now is part of the Apache Software Foundation.

The platform allows scaling to multiple nodes of a cluster, allowing us to easily support the increase in the number of patients and in the volume of data per patient (especially when new signals will be acquired from medical equipment). In addition, it is tolerant to failures, an essential property for a critical application of the health field.

The interaction with Kafka is carried out through subscriptions/publications of "streams of records" (representing data streams). Thus, there are producers who make publications, to send streams of records to the platform, and consumers who make subscriptions to receive streams of records from the platform. This is exemplified in figure 8. Specifically for our prototype, the Data Acquisition module is a producer, Apache Flink is both consumer and producer, and Cassandra and the web server are consumers.

Each record (of a stream) consists of a key and a value. The streams of records are stored in categories called topics. For each topic there is a log, which stores the records of the topic. A topic can subscribe zero, one or more consumers, who will read the desired records (the most recent or past) from a single shared log. The logs are maintained persistently, and are deleted after a specified time of life (whether their records have been consumed or not). The use of writing in the filesystem does not involve a loss of performance because Kafka has pagecache-centric design.

A log can be partitioned. Each partition can be stored in a different node of the cluster, and a partition will only be in one node. Partitioning allows to use the storage of more than one node for the same log. In addition, it allows to increase the performance of the system by means of parallel access to the log (from multiple nodes). Kafka only provides a total order over records within a partition, not between different partitions in a topic. A global order of topic records can be achieved using a single partition topic. However, if the use of multiple partitions are required, a solution can be found by determining which records are assigned to each partition (based on the key) at the producer.

Kafka replicates its partitions over multiple nodes for fault-tolerance. Each partition has one node acting as leader and zero or more nodes acting as followers. The leader handles all read and write requests for the

Topic	Key	Value
Alerts	<patient_id>	<time_sec>, <risk_coefficient>, <nurse_alert>, <nurse_treatement>, <physician_alert>, <physician_treatement>
ECG_1	<time_msec>	<sample>
VitalSigns_1	HR	<sample>
VitalSigns_1	RR	<sample>
⋮	⋮	⋮
ECG_*n*	<time_msec>	<sample>
VitalSigns_*n*	HR	<sample>
VitalSigns_*n*	RR	<sample>

Table 1: Kafka topics of our prototype

partition. At the same time, the followers passively replicate the leader. If the leader fails, one of the followers will automatically become the leader. For load balance purpose, each server acts as a leader for some of its partitions and a follower for others.

In our prototype, on the one hand, a unique alert topic has been created for all patients. Thus, each consumer of alerts (the web server and Cassandra) will make a single subscription to Kafka to receive all alerts, whatever the patient. On the other hand, each patient is identified with a number ranging from 1 to the total of possible inpatients. The following topics are defined per patient: a topic for the ECG signal and a topic for the Vital Signs. In table 1 the content of each topic implemented in our prototype is shown, where *n* is the maximum patient *id*, HR is the heart rate and RR is the respiratory rate. All topics have been defined with a single partition. This allows to preserve, in a simple way and without any detriment, the global order of the records of the streams.

The criterion for determining topics for data coming from the Data Acquisition module, is as follows. A topic groups different parameters when two conditions occur: the measurements of the parameters are made at low frequency, and normally the parameters are required together (by Apache Flink, Cassandra, or the web server).

4.2 Stream Data Processing

Stream Data Processing performs two activities: signal analysis and medical rule processing. This module is implemented by Apache Flink [12], a stream processing framework to create distributed, scalable, low latency, and fault tolerant applications. Other frameworks offer similar solutions but using microbatching techniques (like the well-known Apache Spark with Spark Streaming). Unlike them, Flink was created with Streaming Processing in mind, allowing the processing of individual elements of a stream with very low latencies.

Flink works only with data in main memory. Therefore, it is necessary that all data fit in this memory. Fortunately, Flink implements its own memory management inside the Java Virtual Machine (JVM), with less garbage collection overhead. Furthermore, Flink can scale to several nodes of a Cluster (or Cloud), allowing the use of more main memory, and the performance improvement through parallel computing.

Flink supports fault tolerance through checkpoint-restart mechanism to consistently recover the state of the distributed streaming dataflow under failures. The checkpoint can be stored in a configurable place, possibly using a distributed file system. In case of a program failure (due to a failure in software, in computer hardware, or in the network), Flink stops the distributed streaming dataflow. The system then restarts from the last successful checkpoint. As our application has a small state, the checkpoint is very light-weight and can be done frequently with low impact on performance. It is necessary that the checkpoint interval of Flink be consistent with the retention time configured for the Kafka logs. In another case, the recovery will not be complete.

Currently, in our prototype, signal analysis is done with an application written in C language and its integration into Apache Flink is under development. So, at this time, we use Appache Flink only for the processing of medical rules.

Basically, Flink programs are composed of the following 3 parts. *Data source* is the incoming data to be processed. *Transformations* is the processing step, that is, the modifications on the incoming data. Finally, *Data sink* is where Flink sends data after processing.

Particularly to our system, each part is performed as following:

Data source: Flink makes a subscription to Kafka for each patient *id* (from 1 to the maximum number of inpatients) in the topics ECG_*id* and VitalSigns_*id*.

Transformations: As data is received from Kafka, Flink analyses if conditions specified in rules (associated with each patient *id*) are met.

Data sink: When conditions of a rule are met, an alert is issued by producing a new record in the Alerts topic of Kafka.

Flink is natively prepared for integration with Kafka, so it is not necessary to add a special connector between both platforms.

Flink supports the kind of processing required for medical rules. Flink's Complex Events Processing library (CEP) is of special interest for our purpose. With it, Flink is able to process information by detecting patterns that occur in the data, also called events, and then to produce some output. The CEP library has a Pattern API. This API provides tools to detect sequences of

patterns that can be extracted from the input stream. This sequence can also be seen as a graph where each node is a simple pattern and the transitions are made through the fulfillment of a specified condition.

4.3 NoSQL Database

During the time that a patient is hospitalised, it is vital to store all your health data (including alerts) to be consulted by nurses and doctors. These data need to be written in secondary storage and without any compression for quick access. Once a patient leaves the inpatient unit (ICU/IMCU), their data are removed from the local system. However, before being eliminated, data need to be stored (possibly compressed) in the Public Cloud. This Historical Data will be used for Knowledge Extraction. In addition, some data may be recorded in the EHR system.

A database is needed to store medical data generated for each patient during his hospitalisation. The high-frequency of signal data (such as the ECG), multiplied by the number of patients, will produce a very high number of insertions in the database. In turn, as they occur, it is required to respond quickly to queries originated by the web server. The NoSQL databases are appropriate for these requirements and, within existing ones, we choose Cassandra [13].

Regarding the data model, Cassandra's philosophy is to create optimised tables for certain queries and to not implement expensive operations such as joins. Instead, it opts for data redundancy. Cassandra works in a distributed manner and is fault-tolerant. Replication in different nodes allows low latency operations.

Cassandra does not have native connection with Kafka. However, Kafka provides Kafka Connect, a means for integration with other systems through the creation of connectors. There are two types of connectors: the Source Connectors, which import data of a system and insert them into one or more topics (acting in a similar way as a producer) and the Sink Connectors, which export Kafka information to a target system. The latter allows Cassandra to be connected with Kafka, and to be updated as data from the Kafta topics (Alerts, and ECG and VitalSigns for each patient) are ingested.

4.4 Web Server

The User Interface module presents patient data to doctors and nurses. The data involves: alerts, signals, vital signs, and any other medical data. The data can be presented in real time, and in that case the data need to be extracted from Kafka. In addition, it may be necessary to present past-time data. For example, a doctor or nurse might check a patient's ECG and temperature curves, which occurred minutes ago, or at night. In this case, data need to be extracted from Cassandra.

	System Node	Support Node
Processor	1 x Intel Xeon E5-2630	1 x Intel Core 2 Quad Q6600
	6 cores, 12 threads	4 cores, 4 threads
Main Memory	16 GB	8 GB
kernel version	Linux Debian 4.9.18-1	Linux Debian 4.9.82-1

Table 2: Characteristics of the experimental platform

The User Interface is implemented by a web server and web or app clients. The server connects to Kafka (using a Kafka API) and subscribes to the topics of interest, to receive data in real time. When it is required to access past time data, the web server queries Cassandra.

The data from the client interfaces should be updated as the server receives data from Kafka. The typical polling technique (in which each required data need to be requested) is not appropriate for this situation. On the contrary, once a client has requested a certain data stream to the web server, data should flow continuously. To carry out this type of client-server communication, the WebSockets protocol (defined in RFC 6455) can be used. This provides full-duplex communication channels over a single TCP connection. Through a channel, the client can make requests or send data to the server. In turn, by another channel, the server can send data to clients, without request for them constantly. In our prototype we use the Socket.IO library [14], an implementation of WebSockets with extra features.

4.5 Experimentation

The objective of the experimentation is to determine if a server of modest characteristics could support the processing for patients at the ICU (with 7 beds) and IMCU (with 5 beds) of the Francisco Lopez Lima Hospital. The prototype implements the Real-time Big Data Infrastructure using a single node.

A support node is used for:

- Emulate the Data Acquisition module: data streams are generated by Python scripts.
- Capture the alerts: a consumer of Kafka, implemented in Python, receives the alerts.
- Take measures for performance evaluation.

The performance evaluation of our prototype consists of determining the minimum, maximum and average latency to issue alerts. The latency time of an alert is measured from the moment the last necessary data that causes the alert is sent by the additional node, until the alert reaches the additional node.

The characteristics of each node used for experimentation are shown in table 2. Both nodes are connected to a local network of 1 Gbps.

Experiments were performed for 8 and 20 patients. For each patient it is generated an ECG signal with a frequency of 1 sample per millisecond, and 5 vital

	8 patients	20 patients
Minimum (msec)	18	29
Average (msec)	154	167
Maximum (msec)	290	323

Table 3: Alert issue latency

signs with a frequency of 1 sample per second. Each experiment runs for 5 minutes, and every 15 seconds an alert per patient occur. Table 3 shows the minimum, average and maximum latency required to issue alerts. The result obtained allows to determine that the prototype, running on a modest server, is suitable for use in the FLLH. However, it is necessary to use more than one node for the system to be faul tolerant.

5 Conclusions and Future Works

Our objective is to develop a computer system that allows the automatic and early detection of deterioration of patients hospitalised in ICUs and IMCUs, through the real-time processing and analysis of digital health data. In this article the challenges and the proposal of solutions that we implemented in a prototype were discussed. The prototype was developed and evaluated to be used in a public hospital of Argentina.

The general problem of ICUs/IMCUs was presented. We have described a high-level system architecture which supports multiple hospitals without Internet leased lines. The solution uses a computing system at each hospital and a Public Cloud, used to store historical data and for knowledge extraction. The difficulty of extracting data from medical equipment using unknown interfaces (hardware and software) was discussed. We have presented a solution based on an embedded system that we develop for acquire the electrocardiogram (ECG) signal from an analog output of a medical monitor, performs an analog-digital conversion, and transmits it via WiFi to the platform that process the signal. We have detailed a real-time Big Data infrastructure that, based on streams of signals and other health data, allows to process rules to determine and issues alerts indicating risk in the health of patients. The infrastructure is distributed, scalable, fault tolerant and interoperable, based on Free Software products.

We believe that experimental results demonstrate the feasibility of the techniques and technologies used, leaving solid foundations for the construction of a reliable system for medical use, able to scale and support an increasing number of patients and captured data.

As future works, different fault tolerance configurations will be evaluated. The detection of QRS complexes of ECG signals is expected to be integrated into the prototype. Furthermore, it is planned to incorporate the detection of anomalies in ECG signals to avoid the contamination of the system with erroneous data. It is necessary to acquire new signals: oxygen saturation in blood, body temperature and blood pressure. Interconnection with mechanical respirators is also of interest. Finally, the research will be directed to the knowledge extraction module, used to define rules for pathologies prediction.

Acknowledgements

We thank Ariel Stancato for his collaboration in the design of the unipolar positive signal generator circuit.

Competing interests

The authors have declared that no competing interests exist.

References

[1] A. Bravi, G. Green, A. Longtin, and A. J. Seely, "Monitoring and identification of sepsis development through a composite measure of heart rate variability," *PLoS One*, vol. 7, no. 9, p. e45666, 2012.

[2] S. Balaji, M. Patil, and C. McGregor, "A cloud based big data based online health analytics for rural nicus and picus in india: Opportunities and challenges," in *Computer-Based Medical Systems (CBMS), 2017 IEEE 30th International Symposium on*, pp. 385–390, IEEE, 2017.

[3] "ehcos smarticu." Available at: `https://www.ehcos.com/productos/ehcos-smarticu/`. Accessed on 2018-06-09.

[4] "Excel medical." Available at: `http://excel-medical.com`. Accessed on 2018-06-09.

[5] B. R. Matam and H. Duncan, "Technical challenges related to implementation of a formula one real time data acquisition and analysis system in a paediatric intensive care unit," *Journal of clinical monitoring and computing*, pp. 1–11, 2017.

[6] "Amara health analytics." Available at: `http://www.amarahealthanalytics.com`. Accessed on 2018-06-09.

[7] J. Antony, "A tablet pc based system for ubiquitous patient monitoring and smart alert generation in an intensive care unit," *International Journal of Computer Applications*, vol. 67, no. 6, 2013.

[8] J. Balladini, C. Rozas, F. E. Frati, N. Vicente, and C. Orlandi, "Big data analytics in intensive

care units: challenges and applicability in an argentinian hospital," *Journal of Computer Science & Technology*, vol. 15, 2015.

[9] P. Kligfield, L. S. Gettes, J. J. Bailey, R. Childers, B. J. Deal, E. W. Hancock, G. van Herpen, J. A. Kors, P. Macfarlane, D. M. Mirvis, O. Pahlm, P. Rautaharju, and G. S. Wagner, "Recommendations for the Standardization and Interpretation of the Electrocardiogram: Part I: The Electrocardiogram and Its Technology: A Scientific Statement From the American Heart Association Electrocardiography and Arrhythmias Committee, Council on Clinical Cardiology; the American College of Cardiology Foundation; and the Heart Rhythm Society Endorsed by the International Society for Computerized Electrocardiology," *Circulation*, vol. 115, no. 10, pp. 1306–1324, 2007.

[10] A. B. M. Moniruzzaman and S. A. Hossain, "Nosql database: New era of databases for big data analytics - classification, characteristics and comparison," *International Journal of Database Theory and Application*, vol. abs/1307.0191, 2013.

[11] "Apache kafka." Available at: `https://kafka.apache.org`. Accessed on 2018-06-09.

[12] "Apache flink: Scalable batch and stream data processing." Available at: `https://flink.apache.org`. Accessed on 2018-06-09.

[13] "Apache cassandra." Available at: `https://cassandra.apache.org`. Accessed on 2018-06-09.

[14] "Socket.io." Available at: `https://socket.io/`. Accessed on 2018-06-09.