# An edge-stream computing infrastructure for real-time analysis of wearable sensors data

Luca Greco[a], Pierluigi Ritrovato[a], Fatos Xhafa[b]

[a] *Dept. of Computer and Electrical Engineering and Applied Mathematics (DIEM), University of Salerno, Italy*
[b] *Dept. de Ciències de la Computació Universitat Politècnica de Catalunya Barcelona, Spain*

## Abstract

The fast development of IoT in general and wearable smart sensors in particular in the context of wellness and healthcare are demanding for definition of specific infrastructure supporting real time data analysis for anomaly detection, event identification, situation awareness just to mention few. The explosion in the development and adoption of these smart wearable sensors has contributed to the definition of the *Internet of Medical Things* (IoMT), which is revolutionizing the way healthcare is tackled worldwide. Data produced by wearable sensors continuously grow and could be spread among clinical centers, hospitals, research labs, yielding to a Big Data management problem. In this paper we propose a technological and architectural solution, based on Open Source big data technologies to perform real-time analysis of wearable sensor data streams. The proposed architecture is composed of four distinct layers: a sensing layer, a pre-processing layer (Raspberry Pi), a cluster processing layer (Kafka's broker and Flink's mini-cluster) and a persistence layer (Cassandra database). A performance evaluation of each layer has been carried out by considering CPU and memory usage for accomplishing a simple anomaly detection task using the REALDISP dataset.

*Keywords:* Internet of things, Edge Computing, Big Data.

## 1. Introduction

Nowadays, the use of information technologies in Healthcare has led to an improved quality of treatments and a reduction of expenses, especially for patients whose diseases can be remotely screened. Thanks to the ever reducing costs in the production of powerful embedded computing systems, Internet of Things (IoT) technologies are spreading rapidly and have become a far-sighted investment to enhance both medical and financial services, considering that most countries spend at least 10% of their GDP in Healthcare [16].

The interest in IoT technologies for healthcare is well motivated by their capability to effectively address monitoring tasks that would alleviate or prevent critical events (for example heart stroke and ischaemic heart disease, which are the most frequent causes of death in the world [18]). The explosion in the development and adoption of smart medical sensors for healthcare has contributed to the definition of the *Internet of Medical Things* (IoMT), which is revolutionizing the way healthcare is tackled worldwide.

In this paper we propose a technological and architectural solution to perform real-time data analysis from wearable sensors. Technically, a wearable sensor is a device that can be worn or mated with human skin to continuously and closely monitor an individual's activity without interrupting or limiting his motion [19]. In 2014 more than 15 million wearable smart devices were sold [20]; in 2015 69% of US citizens made use of sensors to track their health status; the same statistics in UK reached 70% [21].

Sensors such as heart-rate monitoring chest strap, accelerometers and gyroscopes are commonly used to measure vital signs or tracking motion, becoming a smart solution in a large variety applications such as fall risk assessment and statistical study on patient's habits. Data produced by this kind of sensors continuously grows (at an overwhelming speed) and could be spread among clinical centers, hospitals and research labs, yielding to a Big Data management problem, since it's naturally characterized by velocity, volume, variety and veracity (due to inconsistency) [10].

IoT technologies in healthcare also benefit from the use of Semantic Web technologies [5] as an effective way to associate meanings to raw, and sometimes apparently useless, data produced by sensors that would be otherwise discarded and unused. The semantic annotation of data streams is carried out using terms available in existing ontology like the MIMU-Wear [6], an extensible ontology that comprehensively describes wearable sensor platforms consisting of mainstream magnetic and inertial measurement units (MIMUs). MIMU-Wear describes the capabilities of MIMUs such as their measurement properties and the characteristics of wearable sensor platforms including their on-body location. As we can see in the experimentation the Semantic annotation process contributes to increase the data size to be analyzed.

The rest of the paper is organized as follows. First, we give some background and report a selection of relevant works about IoT technologies solutions for e-Health (Section 2). Then, in Section 3, we describe our layered architecture, giving some details about each block. Section 4 presents results coming from an evaluation of the system and, finally, the Section 5 contains some final considerations on the proposed architecture and indicates future works.

## 2. Related works and background

The idea of remotely monitoring patients' health status from wearable sensors has been subject of investigation from the scientific community for more than ten years. First solutions involved PC based stations like in [23], where a system to remotely monitor patients using data from ECG and accelerometers is described: such application would report to clinicians periods of elevated heart rate and filter expected critical situations. Nowadays, the interest towards technological solutions enhancing healthcare provision has substantially grown and the maturity of IoT technologies has led to the production of several solutions involving IoT-based healthcare network architectures and platforms. In fact, IoT systems prove to address effectively healthcare at different levels: pediatric and elderly care, chronic disease supervision, private health and fitness management [11].

Most works in literature explore the advances in the main enabling technologies for the Internet of Medical things domain, that include cloud computing (ubiquitous access to shared resources), grid computing or cluster computing, Big Data analytics (increase the efficiency of relevant health diagnosis), communication networks, ambient intelligence (learn human behaviour), augmented reality (surgery and remote monitoring), use of wearable medical devices, but also energy consumption models for cooperative transmission strategies [9]. Typical solutions involve the use of powerful micro-controllers and embedded systems to collect data from wearable sensors. Among the available devices on the market, *Raspberry Pi* and *Arduino* are often preferred choices because of their price and versatility.

A large number of research works of the last years concentrate on systems aimed at sensing the main physiological parameters of the human body. For example, Orha *et al.* [26] present a system, based on an Arduino Uno microcontroller, capable of recording body temperature, blood pressure, respiratory rate, electrocardiogram (ECG) and skin resistance from specialized sensors. In the same way, Yakut *et al.*[24] propose a solution to measure physiological ECG data using a Raspberry Pi and a e-health sensor platform and save ECG data as a text file to the SD-card to allow further processing in a Matlab environment.

Maga *et al.* [25] present a system based on Wireless Sensor Networks technology capable of monitoring heart rate and motion rate of seniors within their homes. The system can remotely alert specialists, caretakers or family members via smartphone about rapid physiological changes due to falls, tachycardia or bradycardia. Another system for patient monitoring and tracking relying on the PANGEA platform is presented by Villarubia *et al.* [7]. This system locates (via RSSI) and monitors users (in particular their cardiac function) allowing a simple TV interaction by means of a Raspberry Pi connected to the Internet. ECG incorporates the necessary hardware to send the data to the system via Bluetooth and allows a basic analysis of the data for sending alerts.

Kaur *et al.* [1] also propose a system for remote monitoring of people pulse rate, body temperature with wearable dedicated sensors by means of a Raspberry PI. Remote Health monitoring is obtained by storing the collected data to Bluemix cloud and allowing the doctor to access (MQTT) and analyze it anywhere, detecting timely any anomaly and showing results in the form of graphs at IBM Watson IoT platform.

A different solution is presented by Alwan *et al.* [13] showing an efficient embedded system based of wireless health care monitoring using ZigBee. The solution involves a data exchange between two systems: an Arduino with ZigBee that sends signals to a Raspberry with ZigBee, that is responsible for measuring patient data and sending it back to the first device. The authors demonstrate on volunteers a case where body temperature is monitored to diagnose fever in the patients. The work in [3] shows a novel signal quality-aware Internet of Things (IoT)-enabled ECG telemetry system for continuous cardiac health monitoring applications: a light-weight ECG SQA method for automatically classifying the acquired ECG signal into acceptable or unacceptable class. In particular, a real-time implementation of proposed IoT-enabled ECG monitoring framework using ECG sensors, Arduino, Android phone, Bluetooth, and cloud server is illustrated, with a validation phase using the ECG signals taken from the MIT-BIH arrhythmia and Physionet challenge databases and the real-time recorded ECG signals under different physical activities.

Recent works address also the problem of secure transmissions of health related data such as the work in [8] whose aim is to build a secure IoT-based healthcare system, operating through the BSN architecture. Robust crypto-primitives are used to ensure transmission confidentiality and provide entity authentication among smart objects, the local processing unit and the backend BSN server. The healthcare system is also implemented with the Raspberry PI platform, demonstrating its practicability and feasibility.

Other works investigate the use of IoT systems for limbs monitoring, such as [14] where Mathur *et al.* propose a low cost mobile sensor platform to monitor patients with prosthetic lower limbs. An Android mobile device captures data from a wireless sensor unit and gives the clinician access to results from the sensors. A Raspberry Pi Zero analyzes data used for remote monitoring the tissue health of lower limb amputees achieved using the Gaussian process technique. On the other hand, Villeneuve *et al.* [12] propose an approach to estimate simplified human limb kinematics, based on measurements from two low-power accelerometers placed only on the forearm. The system is considered in a Bayesian framework, with a linear-Gaussian transition model with hard boundaries and a nonlinear-Gaussian observation model. The approach proves that arm kinematics can be estimated from only two accelerometers on the wrist and this is a crucial step toward in machine monitoring of users health and activity on a daily basis.

Another interesting application is described by Chen *et al.* in [2], where a trust-based approach for information sharing in a health Internet of Things (IoT) system is proposed. Smart IoT devices share location-based information obtained through their personal area networks (PANs) with the goal of maximizing the safety of their human owners( e.g. a pollutant sensitive user must determine whether or not he/she should enter a location at a particular time to avoid health related issues). There are also applications in voice pathology monitoring: the work of Muhammad *et al.* [4] discuss the feasibility of a voice pathology detection system using a local binary pattern on a Mel-spectrum representation of the voice signal and an extreme learning machine classifier to detect the pathology. Although many works have been recently proposed in the field of health monitoring, most proposals focus on the analysis of data related to a single individual or small communities without investigating the implications of an extension of such solutions to large numbers of users. Our aim is to propose a global approach that does not focus on the individual patient but extends to a wider context (such as medical structures, hospitals...), also allowing to improve research and practices in the Healthcare domain (for example by sharing statistics about effectiveness of drugs and treatments).

## 3. The proposed system

In this paper, our focus is on cluster computing methods and technologies that allow to reveal anomalies within data streams obtained from wearable devices for e-health purposes. In this context, particular attention has been given to online real-time processing.

Since the wide diffusion of personal care systems prevents to define a standard way to handle health data and its digitalization process, we adopt semantic web standards for data representation and propose a general mixed architecture composed of embedded devices and cluster infrastructures. This solution allows to deal with the huge amount of data streams expected to be generated in a real medical scenario, trying to minimize data loss and guarantee continuous availability required by critical applications.

To make the system reliable and prone to future re-factoring and extensions, we propose a modular architecture where resources are distributed to handle complex tasks.

In particular, we explore the feasibility of a cluster infrastructure that could potentially be adapted to fit the requirements of hospitals, clinics and medical organizations and allow improvements the effectiveness of diagnosis and treatments. Then, our solution should be meant as a technologically forefront instrument that allows clinicians to obtain more meaningful data in shorter times.

To achieve the identified objectives, the system must be able to continuously fetch data coming from a number of wearable sensors (which send data independently) attached to patients and analyze the resulting data streams, allowing a persistent storage. The data has to be replicated through the cluster and it is critical to ensure a proper reaction in case of hardware failure, also with minimal data loss. Furthermore, the system must be capable of detecting potential anomalies in almost real-time in order to predict and identify emergencies.

We propose an architecture, depicted in Figure:1, composed of four distinct layers:

### *3.1. Sensing layer*

This layer is responsible for feeding the entire system. It represents the patient-closest layer and is composed by several e-Health wearable devices.

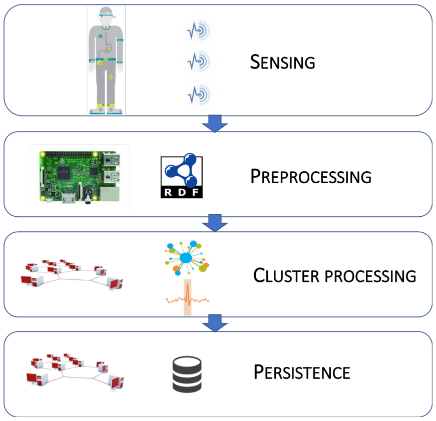

Figure 1: Our layered model.

Depending on the specific sensor, different technologies can be adopted (wired or wireless) to fetch and dispatch data.

### *3.2. Pre-processing layer*

This layer is responsible for retrieving data produced by wearables. Here, an unbounded data stream coming from the *Sensing layer* is collected and converted from raw representation into RDFStreams. Finally, the converted streams are sent to the cluster infrastructure for further processing and analysis.

We chose Raspberry Pi 3 as embedded board due to its large usage in the IoT applications and its low cost. It hosts a Linux OS and a *Node-Red* server providing several virtual computing nodes. We exploited the facilities offered by *Mosquitto* nodes to retrieve data coming from the *Sensing layer*. In particular, the use of the MQTT broker has been crucial to maintain a technology independent communication between wearables and the board; a temporary memory is also available to retain and retrieve data in case of network lacks or next layers' failures.

In Fig.2 we report an example of the designed *Node-Red* programming flow, where the first node represents a *Mosquitto* consumer that fetches data from the previous layer and encodes them in a simple JSON format to sent out towards the other nodes. The second and third nodes are responsible for parsing and converting JSON messages into JSON-LD (standard employed for RDFStreams). Inspired by the TripleWave framework [27], we chose to adopt W3C recommendations to represent semantic data streams and convert data. To dispatch converted data to the next layer, we employ a revised version of the *node-red-contrib-kafka-node* [28].

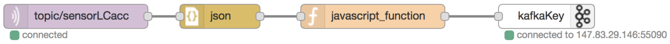

Figure 2: Node-Red programming flow example.

### *3.3. Cluster processing layer*

This layer is responsible for collecting semantic-enriched data and detecting anomalous pattern within such data. In particular, we have two main modules at this stage: the first one is the "hub" of the application, which must buffer data to be further analyzed; the second one has to run in real-time an anomaly detection algorithm to reveal critical situations or emergencies from the acquired data.

Apache Kafka has been chosen to perform the task assigned to the first module since it is a common solution for implementing a Big Data Messaging Hub thanks to its scalability, fault tolerance and high performances [29]. The Kafka cluster has 3 brokers: this number is usually considered an appropriate replication factor (from here RF) [29]. Brokers are identical and each one hosts a number of Kafka topics depending on the number of types of sensors used in the system. This allows all data belonging to a specific sensor to fall in the same topic and every consumers can retrieve the right data just indicating the appropriate sensor. Kafka is actually a bridge between the *Preprocessing layer* and the processing cluster, providing an access point for external systems to consume semantic data.

The second module is devoted to the stream processing of semantic-enriched data coming from *Preprocessing layer* and collected by the Kafka module. Among the stream processors available, the following were considered for the experimentation: Apache Spark, Apache Storm and Apache Flink (Spark and Fink being both batch and stream processors). Spark implements a technique called "micro-batching" which consists of dividing data streams into small chunks and performing a quick batch processing on them. Both Storm and Flink provide true real-time processing and achieve great results in terms of latencies and throughputs. Our choice for the task assigned to the second cluster was Apache Flink thanks to the higher-level API and better results in benchmarks exposed in [30] [31].

The Flink cluster is responsible for getting data from Kafka cluster and processing it to detect potential anomalies. A distributed version of Hierarchical Temporal Memory (HTM) algorithm [32] has been implemented on Flink to address the problem of anomaly detection. We provide some details about the HTM distributed implementation in Section 3.5.

Since medical values strongly depend on the time they are generated, we associate timestamps to each physiological value (Flink Event time logic) to allow the anomaly detection algorithm process data in chronological order.

Due to network lacks or processing latencies, there could be delays in delivering messages to Flink. We set a fixed upper threshold for the delay to allow Flink re-insert each delayed message in the stream correctly: if the delay exceeds such threshold, the message is automatically discarded. Technically, such operation introduces an additional delay, since it is not natively provided by Flink. Anyway, it ensures a more accurate anomaly detection limiting the number of out of order messages.

### *3.4. Persistence layer*

This layer is responsible for storing data analyzed by the *Cluster processing layer* to allow further analysis. It is also an access point for external systems to retrieve stored data. For this task, we compared two common solutions of NoSQL database management systems: Apache Cassandra and Apache HBase. Both are column-oriented databases and can provide distributed storages but they show some differences in performance and offered facilities. Cassandra exposes a query language, called CQL, which is very similar to SQL, enabling easy migrations for accomplishing advanced reporting functionalities. It also offers a greater flexibility than HBase in terms of consistency control [33] and allows a greater number of operations executed per second in loading process contexts [34]. Since Cassandra has proven to perform better than HBase in scenarios with balanced number of writes and reads [35], it was our choice for implementing the Persistence Layer.

Actually, we need a P2P architecture that avoids single points of failures and reaches high availability, since our cluster has a ring topology (with no masters). Cassandra is a good choice also from this point of view: inspired by [37], we adopt a redundancy factor of 3 with the data replication strategy depending on the particular scenario. Since our system is meant for a group of clinics or medical institutions (with many hospitals) a cluster geographically spread into multiple racks seems a realistic solution.

We defined two NoSQL tables to achieve the best performance in retrieving data [37]. The first one collects all detected values, the code of the patient, the name of the specific sensor, the observed parameter, the timestamp and a verbose representation of the related RDFStream. The second table is devoted to collecting anomalous values: here an additional field that indicates the anomaly index of the specific value is added to the ones present in the previous table. The *compound partition key* for both tables is the pair patient-sensor while the *clustering key* is the timestamp.

### *3.5. Hierarchical Temporal Memory algorithm*

A general definition of *anomaly* is a point in time where the behaviour of a system is unusual and significantly different from the past [38]. In our context, we can consider an anomaly as an unusual value (or sequence of values) in a continuous data stream. The Hierarchical Temporal Memory algorithm is considered a foundational technology for the future of machine intelligence. It is inspired to the biology function of the neocortex and implements continuous unsupervised learning so it does not need a training step on data [39]. Moreover, it can be applied to almost every kind of data.

In Fig. 3 we provide a simple schema illustrating the HTM process. Considering $X_t$ the current input of the system, HTM will compute two values: $a(X_t)$ and $\pi(X_t)$. The former is a sparse binary code representation of the current value while the latter is a vector which represents a prediction of the $a$ function for the future input. Using $a(X_t)$ and $\pi(X_t)$ the algorithm evaluates a first raw anomaly score with the following equation:

$$s_t = 1 - \frac{\pi(x_{t-1}) \cdot a(x_t)}{|a(x_t)|}$$

*St* represents a 0 to 1 constrained value which conveys how much the current input is predicted, in particular 0 means fully predicted and 1 is unpredicted. Raw anomaly scores and involved functions are computed every time a new value arrives as input of the system. In order to detect anomalies another step is required: a raw anomaly score is just a predictive parameter which does not represent a reliable way to describe anomalies. Sometimes having a spike or out-of-bound values in data flow is absolutely normal so to obtain a useful information we have to apply a threshold method to the raw anomaly score. Therefore, a real anomaly likelihood can be evaluated considering a window of the last $n$-calculated raw score and computing a normal distribution with the following average and variance:

$$\mu_t = \frac{\sum_{i=0}^{i=W-1} s_{t-1}}{k}$$

$$\sigma_t^2 = \frac{\sum_{i=0}^{i=W-1} (s_{t-i} - \mu_t)^2}{k-1}$$

A threshold is applied to the Gaussian tail probability in order to decide if it is necessary to raise or not an alarm. So, the final anomaly likelihood is defined as the complement of the tail probability $L_t$:

$$L_t = 1 - Q(\frac{\widetilde{\mu}_t - \mu_t}{\sigma_t})$$

It is interesting that in a noisy scenario variance will be large and a spike in values flow has no great impact on anomaly likelihood score: accordingly to the noisy nature of the case, instead a series of abnormal value influences $L_t$ score and highlights an anomaly in the observed system's behaviour. Finally anomalies can be detected thresholding the $L_t$ score, triggering a particular event or alarm depending on the application. Ahamad *et al.* describe more in detail use cases of HTM algorithms [32].

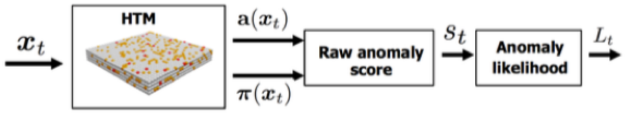

Figure 3: HTM anomaly detection algorithm schema [32].

A distributed version of this algorithm, called Flink-HTM [40] was employed to use HTM on the Flink cluster. The anomaly detection algorithm actually requires a neural network to be executed. The construction of such a network is strongly related to the data it has to analyze. We adopted a standard template (used in most cases by the Numenta engineer) that we customized to fit the provided data patterns. Particular attention was given to the network resolution: a more accurate output implies a greater delay in executing the analysis. The resolution strictly deals with how much regular values can typically differ from each other: very close values require a more fine-grained resolution to detect anomalies. A threshold also needs to be identified in order to distinguish anomalies from regular values.

## 4. Experimental study

For testing our architecture, we used the *REALDISP* [15] dataset as a realistic datasource to feed the system. It contains about 7 GB of data in the form of log files from wearable sensors placed on different parts of 17 individuals. Data have been captured at 50 Hz rate. Each record contains information about time when data was collected. The wearable sensors set comprises accelerometers, gyroscopes and magnetometers.

The hardware used for experimentations and tests was provided by *Research and Development Laboratory (RDLab)* [41] of Computer Science Department at UPC BarcelonaTech. The physical machines employ Intel(R) Xeon(R) CPU X5550 2.66GHz octa-core. Each node hosts an instance of Ubuntu 12.04.2 LTS. The infrastructure is composed of 5 nodes: a node with a single-core CPU and 4 GB of RAM hosts a Kafka broker, a cluster of 3 identical nodes with dual-core CPU and 4 GB of RAM runs a Flink instance and finally a node with a dual-core CPU and 2 GB of RAM hosts a Cassandra broker. Finally, to implement the *Preprocessing layer* a Raspberry Pi 3 is used.

### *4.1. Nodes performance*

As explained in the previous section, our system is composed of four distinct layers: a sensing layer, a pre-processing layer (Raspberry Pi), a cluster processing layer (Kafka's broker and Flink's mini-cluster) and a persistence layer (Cassandra database).

Here we want to provide a performance analysis for each node, in terms of cpu and memory usage during tasks execution. Our analysis is carried out at different ingestion rates in order to better evaluate system behavior.

#### *4.1.1. 50 Hz ingestion rate*

A first experiment set an ingestion rate of 50 Hz which is typically a high value for medical sensors (usually reaching 20-25 Hz rate). Such a rate determined a time limit for the experiment.

|  | **INPUT** | **OUTPUT** |
|---|---|---|
| Raspberry | 45 KB/s | 400 KB/s |
| Kafka | 400 KB/s | 1.2 MB/s |
| Flink | >1.2 MB/s | 1.2 MB/s |
| Cassandra | >1.2 MB/s | n.d |

Table 1: *Input and output produced throughputs with an ingestion frequency of 50 Hz.*

In table 1 the throughputs produced for each software are illustrated while in Fig.4 we report performance of the pre-processing layer (Raspberry Pi) by displaying the quad-core cpu usage and the memory amount required to manage the process.

In particular, when the system starts, the Raspberry Pi runs an instance of a *Mosquitto* broker with 8 topics and the related consumers, 8 running javascript independent functions and 8 Kafka producers which send keyed messages to 8 different Kafka topics. In order to get most reliable results and to avoid to affect statistics, all non-essential interfaces and services like bluetooth, GPIO, Serial and others were deactivated. Data originated from the scripts (which simulate the sensors) consists of 8 sequences of messages of 113 bytes sent with a frequency of 50Hz each one.

After a first idle situation, at sec 10 the *Mosquitto* broker and the *Node-Red* server processes start, determining an initial peak that quickly decays. At sec 53, when the data stream is sent and starts to be elaborated by the Raspberry Pi, we observe a strong rise in cpu usage, up to around 60%.

The Flink's cluster is composed of 3 nodes having the same configuration with a dual-core cpu, 2 GB of RAM and a disk space of 200 GB. Generally, Flink nodes can cover two roles: Job Manager(JM), which manages and distributes the job and Task Manager(TM). Having a node covering both roles is unusual in Flink due to the required resource sharing, but in our case we observed that configuring a node as a Job and Task manager achieved a good performance level. Particular attention was payed to parameter tuning: heap size, slot number and parallelism for each TM. We tested different configurations to find the best one. Flink runs operators and user-defined functions inside the Task Manager JVM, so the heap amount reserved for each TM should be as large as possible to get more benefits. Clearly, memory is shared with other OS processes so an analysis about memory usage by the Flink application seemed necessary. Some experiments with an increasing

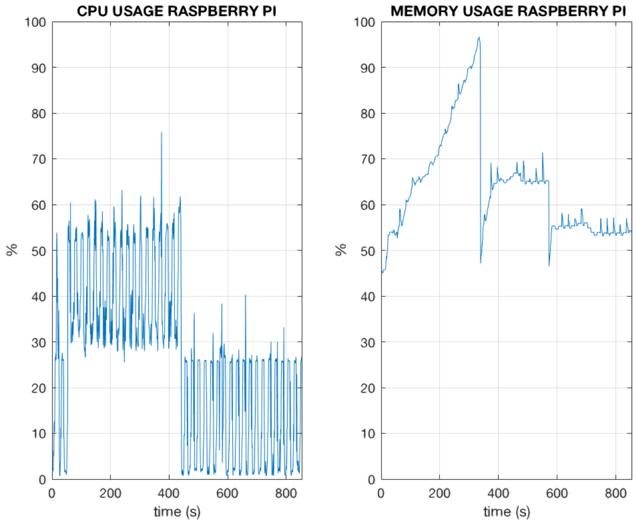

Figure 4: *The left side shows the cpu utilization in a task of around 14 minutes while on the right we can see the memory consumption. On the y-axes there is the usage percentage while on the x-axes the time is expressed in seconds.*

amount for heap were carried out to discover the reachable limit. At the end, we chose heap sizes for each node as shown in table 2. For `node-1` memory has been partitioned between the Job Manager and the Task Manager, reserving a greater amount to the latter.

The first experiment was initially ran using 2 slots each TM, since each TM is equipped with a dual-core cpu. All the 8 sensor streams ingested by Kafka were analyzed. Unfortunately, the TMs were continuously failing due to memory overflow errors; data was ingested too fast for the neural networks to process it and reveal anomalies in time. The reduction of the heap portion

**FLINK CONFIGURATION**

| NODE | ROLE | HEAP (MB) | N. SLOTS |
|---|---|---|---|
| node-1 | JobManager | 256 | Not defined |
| node-1 | TaskManager | 1256 | 1 |
| node-2 | TaskManager | 1512 | 1 |
| node-3 | TaskManager | 1512 | 1 |

Table 2: *The configuration chosen for each Flink node*

allocated for Flink's internal operations from the default value of 70% to 20% was still not enough. Another attempt concerned the distribution of job's tasks within the cluster. Usually, Flink distributes tasks through the nodes trying to maximize efficiency and it tends to allocate in the same slot operators which share data or with similar task; in our case this behaviour could lead to an unbalanced cluster. We set the task distribution strategy with just 1 slot per Task Manager and forcing the application to reserve a specific slot for particular operators (*Slot Sharing Group*).

Therefore, to figure out the limits of the configuration, we did an experiment with just 1 sensor (i.e. 3 simultaneous streams), the accelerometer located on the left calf of an individual. Thus, each Task Manager was deployed with a single slot and the SSG was adopted to have a balanced cluster. In order to implement the analysis of 3 streams, 2 custom slots were specified to host respectively two of the network operators while the third one was deployed by Flink in the *Default Slot* with all remaining operators.

Since two of the three slots are reserved, the other Flink operators are constrained to reside in the *Default Slot* with no possibilities to use parallelism, that would require a separate slot for each parallel instance. Although the parallelism would provide a relevant performance boost, it requires TMs to have more available slots or eventually employing more powerful TMs to avoid the use of SSG.

In Figures 5 and in 6 the CPU usage percentages for the Flink nodes are displayed: we can note that after an initial phase with high CPU usage (job submission), the percentage hardly exceeds the 50%. The sudden low usage period at 330 sec is due to a network lack which prevented the application to consume data from Kafka; this insight is confirmed by the subsequent peak showed in the graph which corresponds to a relative large number of data to analyze. Anyway, this event confirms that Flink and Kafka address successfully a network lack issue. Finally, the elaboration as a whole is performed in real-time since the CPU percentage drops exactly when the raspberry stops to send data.

The Apache Kafka broker was deployed on a node equipped with a single core CPU, 4 GB of RAM and a 200 GB of mass storage. The broker provides 8 topics and each one is divided in 10 partitions. In Fig. 7 the CPU percentage usage is represented; moreover, in Fig. 8 a view of RAM usage and disk space depletion is also provided.

Naturally, the throughput of the test depends on the number of Raspberry units involved. Our results show that this Kafka configuration could handle

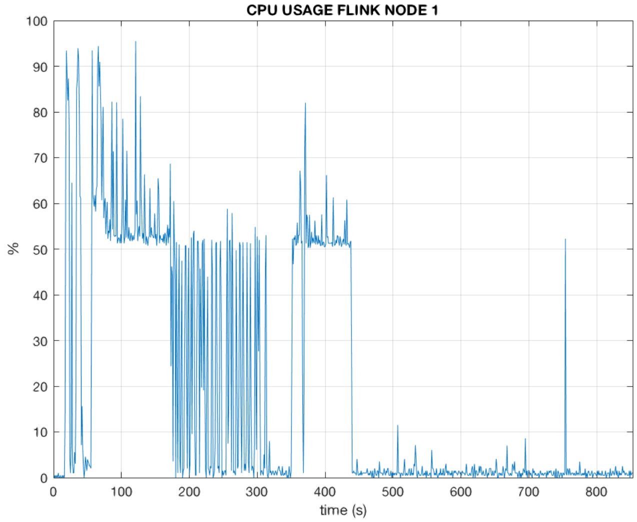

Figure 5: *The CPU usage for the* `node-1` *which hosts both Job Manager and Task Manager*

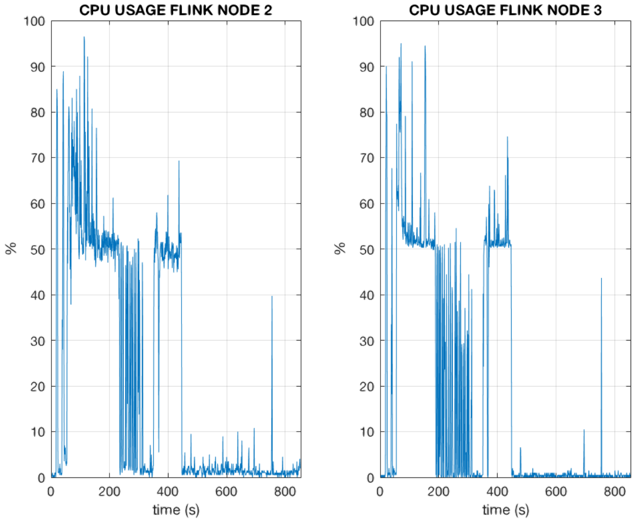

Figure 6: *The CPU usage for the other Flink nodes.*

several Raspberry units easily since the CPU usage remains averagely on a low percentage except for spikes due to a network congestion. RAM usage is quite stable on a certain level since Kafka engine stores incoming data directly on the mass storage, writing them sequentially. Moreover, since

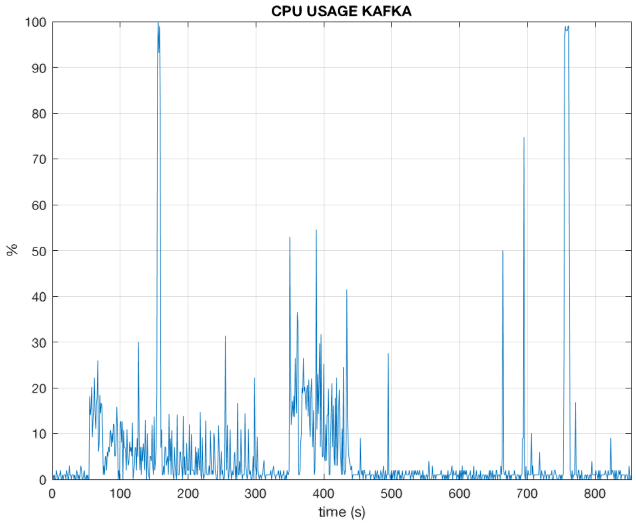

Figure 7: *The Kafka CPU usage. On the y-axe the usage percentage while on the x-axe the time is expressed in seconds.*

data on Kafka is almost never deleted, the filesystem is not fragmented and reads are mostly executed sequentially with high rates.

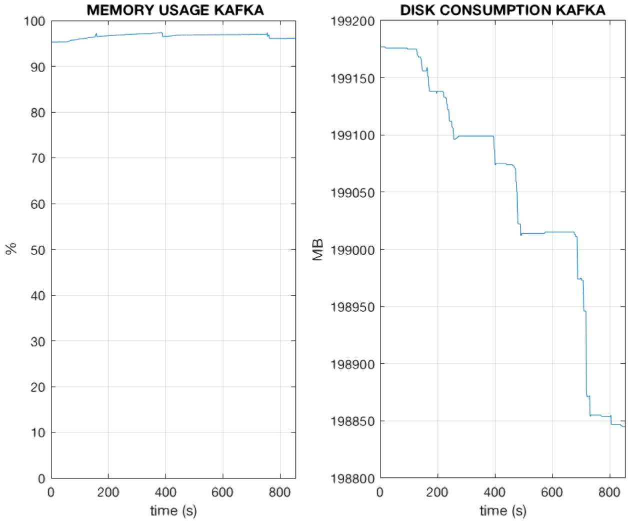

Figure 8: *The RAM memory employment (on the left) and the disk space depletion (on the right). On the y-axe of the rightmost figure the disk amount is expressed in MB while on the x-axe the time is expressed in seconds.*

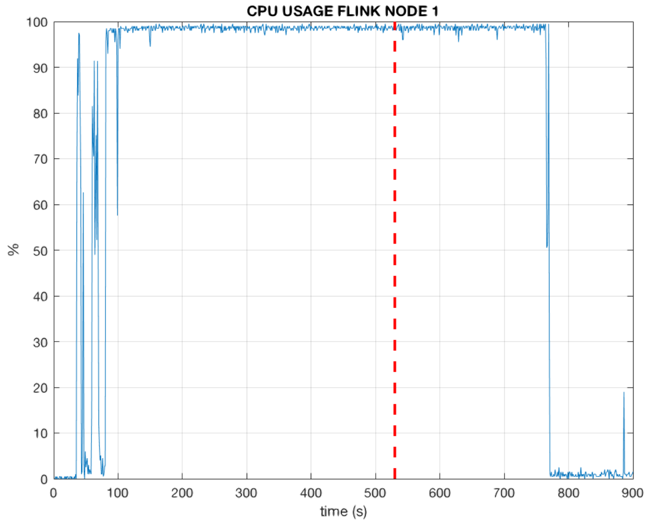

Figure 9: *The CPU usage of the Job/Task Manager of the Flink cluster with 6 streams ingested towards it. On x-axe the time is expressed in seconds. The dashed vertical line represents the instant when the streams end.*

The total disk space is 200 GB and the graph in Fig. 8 illustrates how the space drops when the data stream arrives to Kafka.

Figures 9 and 10 show the performance of Flink nodes receiving data from 2 sensors, that means analyzing 6 streams simultaneously. The heavy usage of CPU immediately stands out.

We observe a very delayed data processing: the vertical dashed line on the graph represents the ending of stream ingestion. Most nodes were not able to perform a real-time analysis and in the worst case, i.e. the `node-2`, the task is completed with a delay of more than 300 seconds. Essentially, data is queued in a long buffer on the network operators not being able to consume in time. Clearly, this is not acceptable in a system where the real-time analysis is essential (real medical scenario) and leads us to consider the ingestion frequency of 50 Hz too much high for handling more than 1 sensor with the adopted infrastructure.

Apache Cassandra (running on a dual-core CPU, 4 GB RAM and a disk of 1 TB) is able to handle a heavier load with respect to the data amount ingested from Flink. Anyway, since it is placed at a lower layer, it is affected by the limitations imposed by the Flink's cluster. Memory usage is around 90% even in idle state, raising to 91% max under load. CPU usage is illustrated

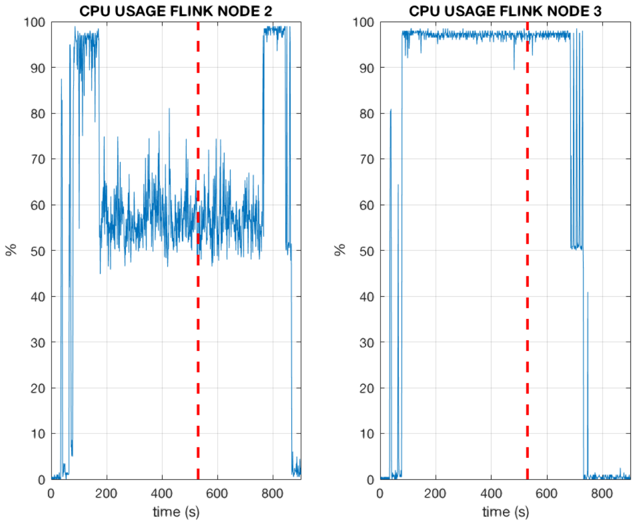

Figure 10: *The 2 Flink nodes and their CPU usage expressed in percentage. On x-axe the time is expressed in seconds. The dashed vertical line represents the time when the streams end.*

in Figure 11: it remains averagely under the 20% in both cases.

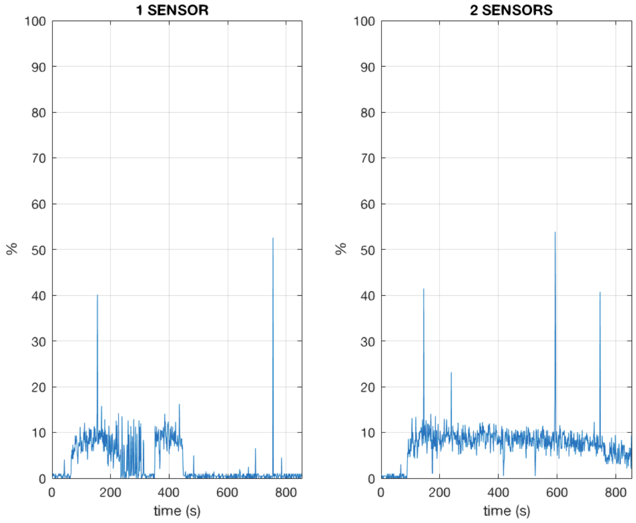

Figure 11: *On the left the CPU usage for the Cassandra node in the first experiment. On the other side the second one is showed. On x-axe the time is expressed in seconds.*

It should be considered that in the second experiment, with the highest

| | **INPUT** | **OUTPUT** |
|---|---|---|
| Raspberry | 23 KB/s | 200 KB/s |
| Kafka | 200 KB/s | 150 KB/s |
| Flink | 150 KB/s | >150 KB/s |
| Cassandra | >150 KB/s | n.d |

Table 3: *Input and output produced throughputs with an ingestion frequency of 25 Hz.*

load for Cassandra, the throughput is nearly 300 KB/s whereas without Flink's limitations it would be at least 1.2 MB/s: these values represent both a load widely bearable by Cassandra.

Finally, we present a statistic about performance with an ingestion frequency of 50 Hz, using a raising number of sensors. Except for Flink, all systems present good performances. Several simulations were done varying the number of sensors: from 2 to 32 for the Raspberry unit and Kafka; from 2 to 64 for Cassandra. Actually, the Raspberry unit showed processing issues (message drop) raising the number of sensors to more than 32. Anyway, the configuration with 32 sensors is an acceptable result considering that it corresponds to an input throughput of 181 KB/s and to an output throughput of 1.6 MB/s. Kafka results are also convincing: the test with 32 producers can simulate a scenario with 4 concurrent Raspberry. For Cassandra, a separate Flink program which simulates just the stream dispatching of the original application was deployed in order to better test the performances of the database. In Fig.12, from the highest to the lowest, the average CPU and RAM memory employment for the Raspberry Pi, the Kafka broker and the Cassandra database are shown.

Raspberry CPU usage presents a slight and gradual increment of few percentage points also among the two higher cases (i.e. 16 and 32 sensors). Nowadays, it is hard to find applications using so many wearables on a single person.

### *4.1.2. Ingestion frequency: 25 Hz*

To better realize the impact of the ingestion frequency on the proposed system, a new experiment sending 25 messages per second was carried out, without changing any of the other parameters. In the Table 3 the input and output throughputs generated with an ingestion frequency of 25 Hz are shown.

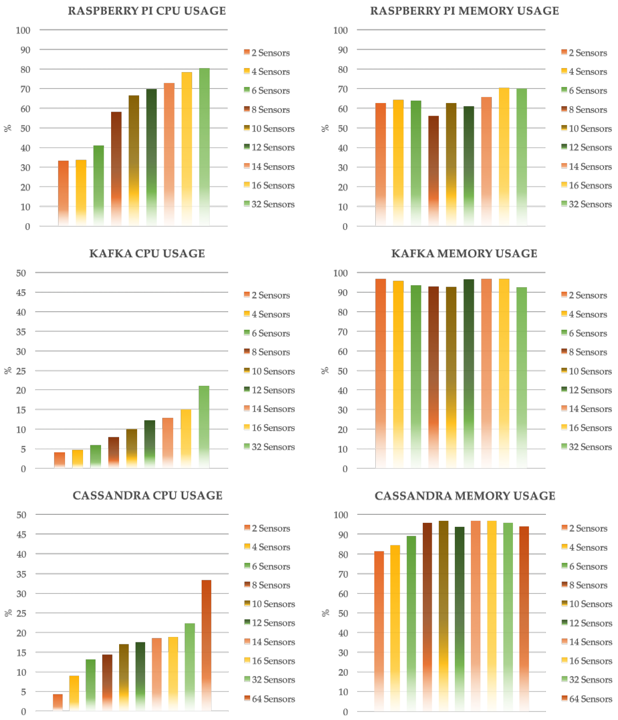

Figure 12: *The bars represent the average values registered experimenting with an increasing number of sensors.*

The objective of this second attempt is to verify if, halving the ingestion frequency, the Flink application is able to analyze 2 sensors simultaneously.

Fig. 13 shows the CPU and memory performance on Raspberry with an ingestion frequency of 25 Hz.

The CPU usage shows only negligible differences while the memory consumption is characterized by a decrease of about 6-7 percentage points. The Kafka graph in Fig. 14 about CPU shows an average usage essentially equal to the previous case but with a more stable trend thanks to the lesser ingestion frequency which avoids sudden back-pressures due to potential network congestions. Flink nodes CPU usage is shown in Fig. 15 and Fig. 16.

Halving the ingestion frequency, CPU usage is still very high and touches the 99%. Even in this case, the job is not completed on time and halving the ingestion rate seems not to be enough to analyze more than one sensor data: anyway, the delay is very reduced compared to the previous case. Finally, a predictable decrease in CPU usage of Apache Cassandra can be observed in

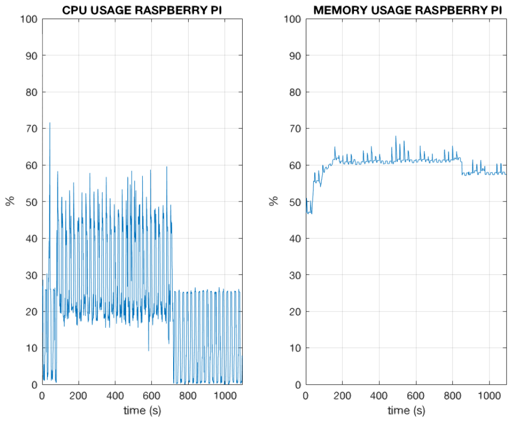

Figure 13: *The CPU and memory usage on Raspberry Pi with an ingestion rate of 25 Hz.*

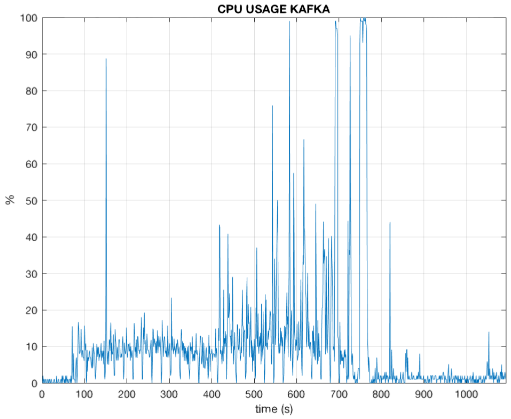

Figure 14: *The CPU graph about Apache Kafka in the second experiments.*

the Fig.17. We also note a memory consumption of about 90-95%.

### *4.1.3. Ingestion frequency: 15 Hz*

The last experiment cuts the original ingestion frequency of 70%. In table 3 we report the input and output throughputs generated with an ingestion frequency of 15 Hz. It should be noted that output values for Flink, Cassandra and Kafka are affected by the limitations provided by the anomaly detection algorithm (at most 2 sensors can be evaluated simultaneously).

The figures 18 - 20 depict the behaviour of Raspberry, Kafka and Cassandra with the new ingestion frequency: here a slight performance improvement

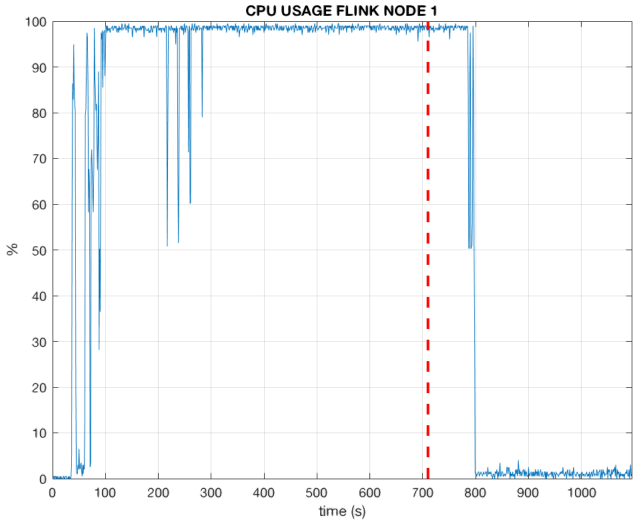

Figure 15: *The CPU graph of the first node of the Flink cluster. The dashed line represents the instant when the stream ends.*

| | **INPUT** | **OUTPUT** |
|---|---|---|
| Raspberry | 14 KB/s | 120 KB/s |
| Kafka | 120 KB/s | 90 KB/s |
| Flink | 90 KB/s | >90 KB/s |
| Cassandra | >90 KB/s | n.d |

Table 4: *Input and output produced throughputs with an ingestion frequency of 15 Hz.*

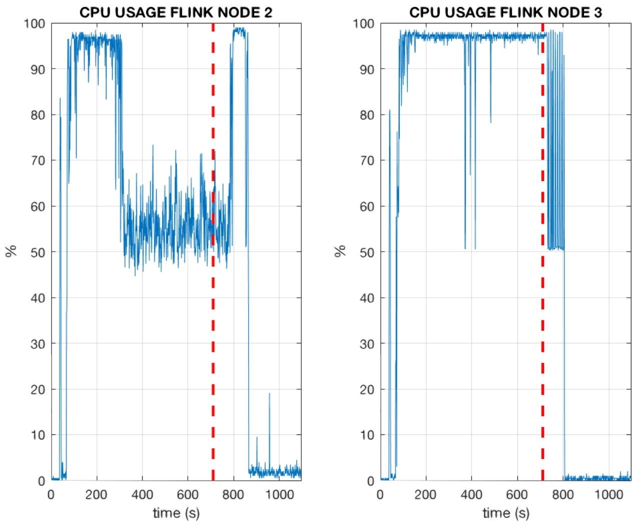

Figure 16: *The CPU graph about the other two Task Managers. The dashed line represents the instant when the stream ends.*

can be highlighted. Moreover, in figures 21 and 22 we can observe how CPU usage for Flink's nodes is reduced, allowing the job to be completed in real-time: the dashed lines now correspond exactly to the end of the computation.

By reducing data rate sent from 300 KB/s to 90 KB/s, we observe some oscillations in Flink CPU graphs since periods of intensive computation are alternated with periods of "silence".

### *4.1.4. Data loss*

During the elaboration, data sent and partially processed could be lost due to different reasons. The most frequent is a network lack but it could also be a dropping Flink. As explained in section 3.3 if a message arrives with a delay greater than a fixed value (800 ms in the implemented case), it is dropped since it could not be added to the stream.

Some statistics about the average data loss are depicted in Table 5. The experiments were executed with the 3 different ingestion rates and, for the two Flink applications, with 3 and 6 data streams. Each test was repeated 3 times for each value. Reasonably, the percentage loss decreases with the lower ingestion rates and analyzing less sensors, thanks to the lower overhead required to run the application.

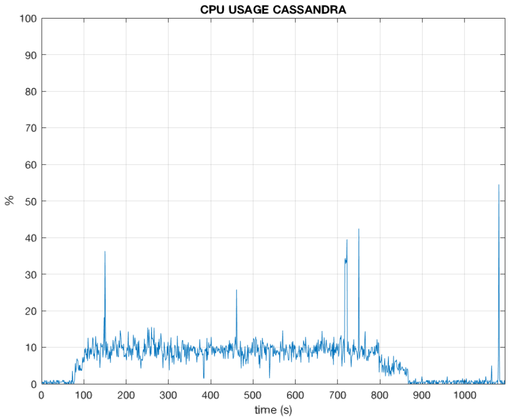

Figure 17: *The CPU graph about the Cassandra database within the second experiment.*

**AVERAGE DATA LOSS (%)**

| **INGESTION FREQ. (Hz)** | **3 STREAMS** | **6 STREAMS** |
|---|---|---|
| 50 | 0.302 ± 0.02 | 0.366 ± 0.03 |
| 25 | 0.284 ± 0.01 | 0.326 ± 0.1 |
| 15 | 0.273 ± 0.01 | 0.293 ± 0.07 |

Table 5: *The average data loss in the system.*

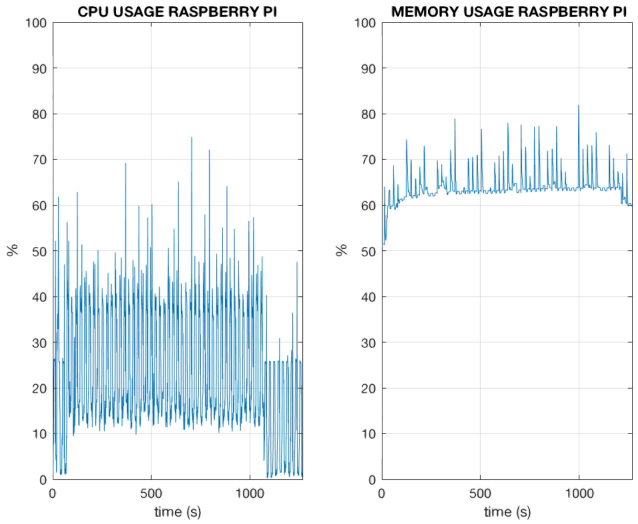

Figure 18: *The graph about Raspberry's CPU and memory consumption with an ingestion rate of 15 Hz.*

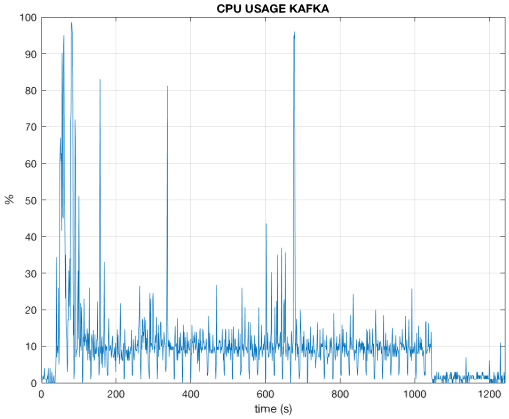

Figure 19: *The graph about Kafka CPU usage with an ingestion rate of 15 Hz.*

## *4.2. HTM results*

Although the aim of our study was to test the performance of the system, not the accuracy of HTM, we tried to set the parameters in order to achieve the best trade-off between accuracy and computation speed. As we explained in a previous section, one of the most important parameters to be tuned is the network resolution. Table 6 sums up the time required to compute the anomaly degree for each record when testing 4 different networks with a

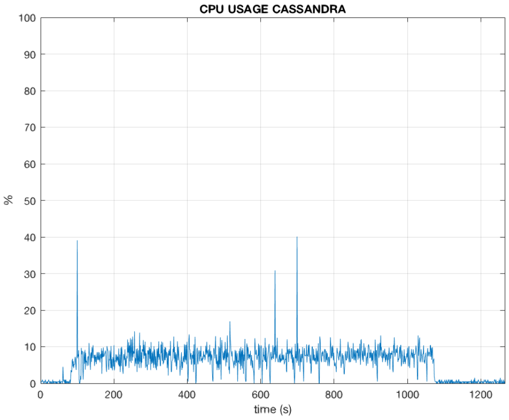

Figure 20: *The CPU graph about the Cassandra database within the third experiment.*

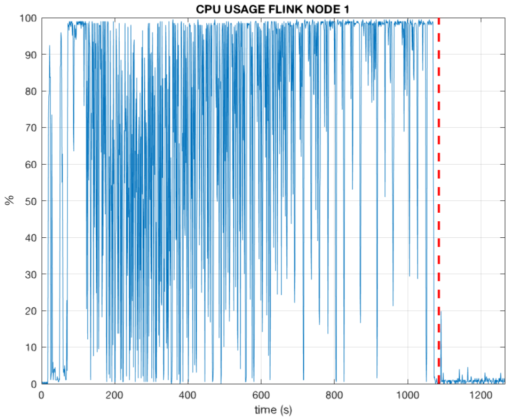

Figure 21: *The usage caused by the job on the first node of the Flink cluster, finally computed in real-time.*

decreasing resolution. Our network creates a set of bins with a size fixed to the resolution value. The algorithm computes a prediction value and puts the obtained value in the correspondent bin. At the next step, the algorithm compares the ”predicted” bin with the destination bin of the real value. If the resolution is too high, the number of bins is very large and many values will fall in bins which do not represent data adequately; moreover, if too few bins are created, wrong predictions will fall in the same bin of the real values leading to false negatives and returning no anomalies. The chosen network has a resolution of 0.3.

The dataset used to represent sensor data belonging to a person is com-

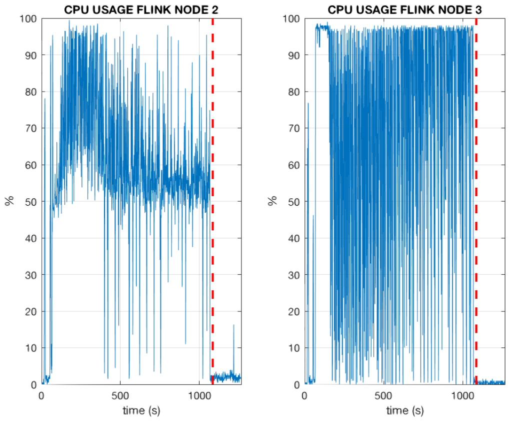

Figure 22: *The usage caused by the job on the 2 last node of the Flink cluster, finally computed in real-time.*

| **RESOLUTION** | **TIME PER RECORD (ms)** |
|---|---|
| 0.1 | 25.98 |
| 0.3 | 16.35 |
| 0.5 | 15.19 |
| 0.7 | 15.17 |

Table 6: *Comparison of the elaboration time required to compute a single record with several network resolutions.*

posed of 180.000 records temporally separated from 20 milliseconds. The values considered here as example match the acceleration along the X-axe. In Fig. 24 and 26 we present a portion of the dataset (for readability), in particular we show the first 20.000 and 50.000 records. We employ a *continuous learning* algorithm: this means that the network does not use a learning phase to perform anomaly detection but takes an initial time period (depending on data variability) to identify the regular data pattern. Indeed, in Fig. 25 and 27, that show respectively the anomalies found in the first 20.000 and 50.000 records, we can observe the presence of a initial phase of *1* values representing strong anomalies. This is due to the initial lack of data knowledge for the network. In this test, after about 6 seconds, the network is ready to evaluate the forthcoming data.

It should be noted that HTM is a memory-based system and has not ability to "understand" data meaning but evaluates repeatability and recurrence of patterns. Therefore, it achieves good results in case of recurrent values. Comparing respectively the pairs in Fig. 24 - 25 and Fig. 26 - 27 we can observe how HTM registers high anomaly degree spikes (values closer to 1) exactly where abnormal peaks in the dataset graph occur. It should be noted that actually HTM do not discard anomalies but calculates an anomaly degree, constrained from 0 to 1. Defining which values represent anomalies and which not is a task heavily dependent by the application. The histogram in Fig.23 depicts the number of anomalies found with 4 networks by varying thresholds. Generally, resolution has to be chosen on data basis, in fact a high number of anomalies does not necessarily correspond to a higher accuracy, because false positives also increase. In this project, a value of 0.8 was chosen as a threshold to distinguish between anomalies (>= 0.8) and standard values (< 0.8). Only values with an anomaly degree greater than 0.8 were stored into Cassandra as anomalies.

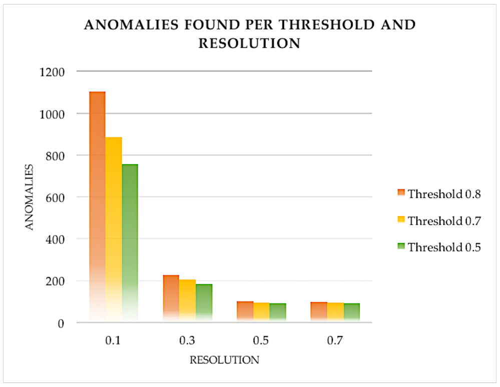

Figure 23: *A summary of the consequences of using a network with different resolution value.*

## 5. Conclusion

In this paper we proposed an architecture for an Internet of Medical Things scenario, where data collected from wearable sensors enable the system to raise alarms or trigger autonomous reactions within few seconds whenever an emergency occur. We employed latest technological frameworks for edge and cluster computing (Apache Flink, Kafka and Cassandra) and tested an HTM algorithm implementation to identify anomalies within data streams. Our experimental stage suggest that all frameworks (except

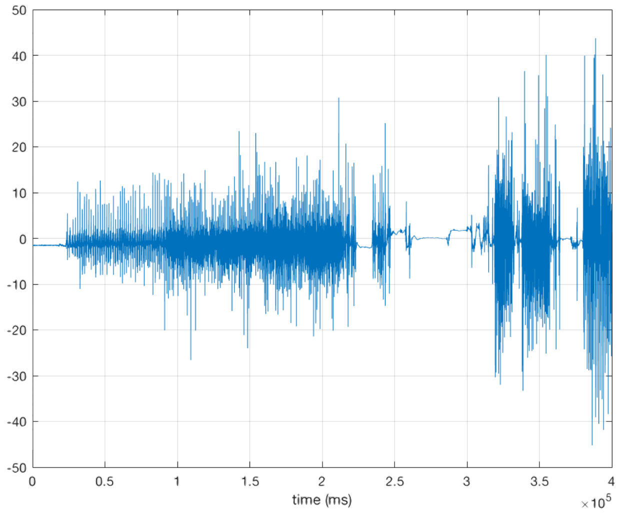

Figure 24: *The first 20.000 records of the dataset.*

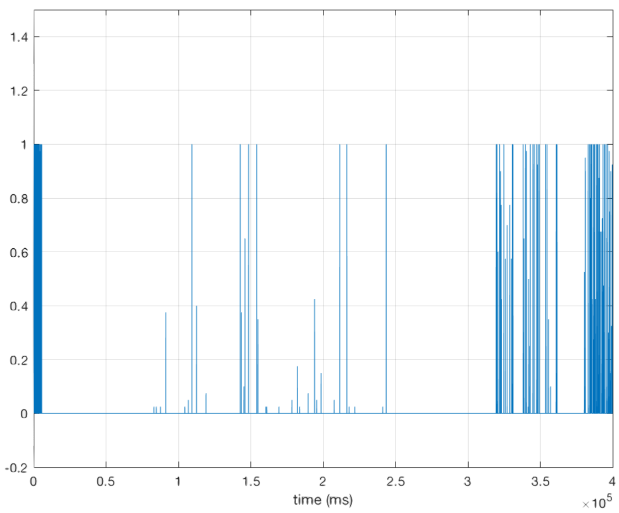

Figure 25: *Anomaly peaks found in the first 20.000.*

Flink) fulfilled the assigned task showing great performance. In our view, the reasons for Flink low performance are mainly related to *Flink-HTM* implementation. Anyway, the architecture provides a good solution concept to implement a real-time anomaly detection system; in a real scenario a more powerful physical infrastructure would be desirable. The use of semantic

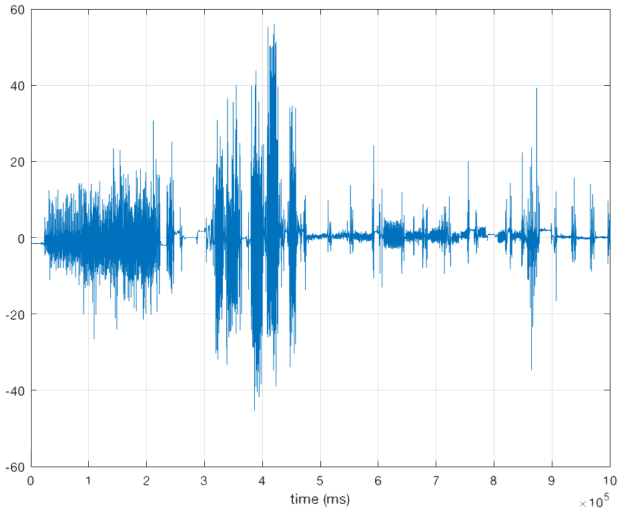

Figure 26: *The first 50.000 records of the dataset.*

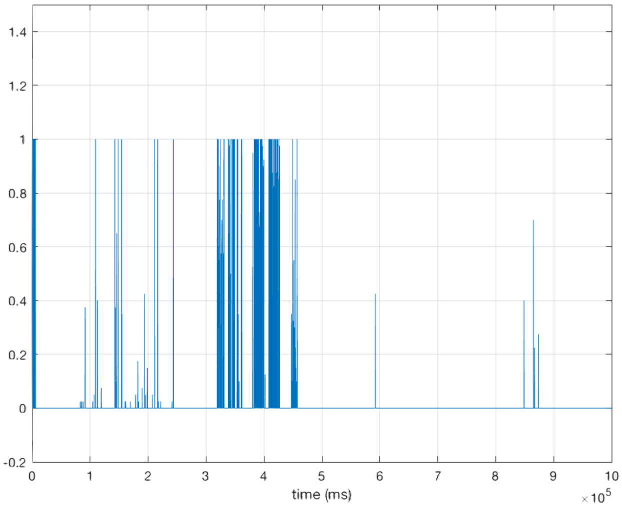

Figure 27: *Anomaly peaks found in the first 50.000 records*

technologies can help distinguishing a real emergency or the symptoms of a disease from a natural change in physiological values.

One of the most important strengths of the architecture is its modularity: it is simple to expand and re-factor layers, replacing frameworks or adding new nodes to the processing clusters without changing the whole configuration.

As a future work, we are interested in implementing and evaluating other anomaly detection algorithms and techniques. Moreover, we mean to focus on non-invasive wearable sensors to use for elderly and dementia patients [17].

## References

[1] A. Kaur and A. Jasuja, *Health monitoring based on IoT using Raspberry PI*, 2017 International Conference on Computing, Communication and Automation (ICCCA), Greater Noida, 2017, pp. 1335-1340.

[2] H. Al-Hamadi and I. R. Chen, *Trust-Based Decision Making for Health IoT Systems*, in IEEE Internet of Things Journal, vol. 4, no. 5, pp. 1408-1419, Oct. 2017.

[3] U. Satija, B. Ramkumar and M. Sabarimalai Manikandan, *Real-Time Signal Quality-Aware ECG Telemetry System for IoT-Based Health Care Monitoring*, in IEEE Internet of Things Journal, vol. 4, no. 3, pp. 815-823, June 2017.

[4] G. Muhammad, S. M. M. Rahman, A. Alelaiwi and A. Alamri, *Smart Health Solution Integrating IoT and Cloud: A Case Study of Voice Pathology Monitoring*, in IEEE Communications Magazine, vol. 55, no. 1, pp. 69-73, January 2017.

[5] Michael Compton, Payam Barnaghi, Luis Bermudez, Raúl García-Castro, Oscar Corcho, Simon Cox, John Graybeal, Manfred Hauswirth, Cory Henson, Arthur Herzog, Vincent Huang, Krzysztof Janowicz, W. David Kelsey, Danh Le Phuoc, Laurent Lefort, Myriam Leggieri, Holger Neuhaus, Andriy Nikolov, Kevin Page, Alexandre Passant, Amit Sheth, Kerry Taylor, *The SSN ontology of the W3C semantic sensor network incubator group*, Web Semantics: Science, Services and Agents on the World Wide Web, Volume 17, 2012, Pages 25-32, ISSN 1570-8268.

[6] Claudia Villalonga, Hector Pomares, Ignacio Rojas, Oresti Baños, *MIMU-Wear: Ontology-based sensor selection for real-world wearable activity recognition*, Neurocomputing, Volume 250, 2016,Pages 76-100, Elsevier.

[7] Villarrubia, Gabriel and Bajo, Javier and De Paz, Juan F. and Corchado, Juan M. *Monitoring and Detection Platform to Prevent Anomalous Situations in Home Care*, Sensors, 14(6): 9900?9921, 2014

[8] K. H. Yeh, *A Secure IoT-Based Healthcare System With Body Sensor Networks*, IEEE Access, Page(s): 10288 - 10299, 2016

[9] Y. Peng and L. Peng, *A Cooperative Transmission Strategy for Body-Area Networks in Healthcare Systems*, IEEE Access, Page(s): 9155 - 9162, 2016

[10] Van-Dai Ta, Chuan-Ming Liu and G. W. Nkabinde, *Big data stream computing in healthcare real-time analytics*, 2016 IEEE International Conference on Cloud Computing and Big Data Analysis (ICCCBDA), Chengdu, 2016, pp. 37-42.

[11] S. M. R. Islam and D. Kwak and M. H. Kabir and M. Hossain and K. S. Kwak, *The Internet of Things for Health Care: A Comprehensive Survey*, IEEE Access, Pages: Page(s): 678 - 708, 2015

[12] E. Villeneuve and W. Harwin and W. Holderbaum and B. Janko and R. S. Sherratt, *Reconstruction of Angular Kinematics From Wrist-Worn Inertial Sensor Data for Smart Home Healthcare*, Page(s): 2351 - 2363, IEEE Access, 2016

[13] O. S. and K. Prahald Rao, *Dedicated real-time monitoring system for health care using ZigBee*, Healthcare Technology Letters,Page(s): 142 - 144, 2017

[14] N. Mathur and G. Paul and J. Irvine and M. Abuhelala and A. Buis and I. Glesk *A Practical Design and Implementation of a Low Cost Platform for Remote Monitoring of Lower Limb Health of Amputees in the Developing World*, IEEE Access, Page(s): 7440 - 7451, 2016

[15] Oresti Baños, Máté Attila Tóth, *Realistic sensor displacement benchmark dataset*, Dataset manual, 2014

[16] World Bank Group IBRD-IDA, Global Health Expenditure database, 2017 *https://data.worldbank.org/indicator/SH.XPD.TOTL.ZS*

[17] Philip Moore, Andrew M. Thomas, George Tadros, Fatos Xhafa, Leonard Barolli: *Detection of the onset of agitation in patients with dementia: real-time monitoring and the application of big-data solutions.* IJSSC 3(3): 136-154 (2013)

[18] World Health Organization Statistics, 2017, *http://www.who.int/mediacentre/factsheets/fs310/en/*

[19] Wei Gao, Sam Emaminejad, Hnin Yin Nyein, Samyuktha Challa, Kevin Chen, Austin Peck et al. *Fully integrated wearable sensor arrays for multiplexed in situ perspiration analysis*, Nature, 2016; 529(7587): 509-14

[20] Shujaat Hussain, Byeong Ho Kang, Sungyoung Lee, *A wearable device base personalized Big Data analysis Model*, Lecture Notes in Computer Science, vol. 8867, 2014, Springer International Publishing Switzerland

[21] Ben Walker, *Every day Big Data statistics*, 2015 *http://www.vcloudnews.com/every-day-big-data-statistics-2-5-quintillion-bytes-of-data-created-daily/*

[22] Tim Conrad, Lydia Ickler, Alexander Reinefeld, Florian Schintke, Robert Schmidtke, Christof Sch?tte et al. *Big Data Analytics in e-Health using Apache Flink and XtreemFS*, Berlin Big Data center, Zuse Institute Berlin, 2017

[23] Chung-Min Chen, Hira Agrawal, Munir Cochinwala, David Rosenblut, *Stream query processing for Healthcare bio-sensor applications*,20th International Conference on Data Engineering, 2004, IEEE

[24] Onder Yakut, Serdar Solak, Emine Dogru Bolat *Measuring ECG Signal Using e-Health Sensor Platform*, International Conference on Chemistry, Biomedical and Environment Engineering (ICCBEE'14), Pages: 65-69, 2014

[25] Magaña-Espinoza P., Aquino-Santos R., Cárdenas-Benitez N., Aguilar-Velasco J., Buenrostro-Segura C., Edwards-Block A. et al. *WiSPH: A Wireless Sensor Network-based Home Care Monitoring System*, Sensors, 2014;14(4): 7096-7119

[26] Ioan Orha, Stefan Oniga, *Automated system for evaluating health status*, Design and technology in Electronic Packaging (SIITME), 2013, IEEE 19th International Symposium for, pp.219-222

[27] Andrea Mauri, Jean-Paul Calbimonte, Daniele Dell'Aglio, Marco Balduini, Marco Brambilla, Emanuele Della Valle et al. *TripleWave: Spreading RDF Streams on the Web*, International Semantic Web Conference, ISWC 2016: The Semantic Web - ISWC 2016, 2016, pp. 140-149

[28] F. Wang, 2016 *https://www.npmjs.com/package/node-red-contrib-kafka-node*

[29] Kafka Official Documentation, 2018 *https://kafka.apache.org* (accessed as of January 2018)

[30] Ellen Friedman, Kostas Tzoumas, *Introduction to Apache Flink*, 2016, OReilly Media

[31] Reza Farivar, Kyle Knusbaum, *Performance Comparison of Streaming Big Data Platforms*, DataWorks Summit/Hadoop Summit, 2016

[32] Subutai Ahmad, Scott Purdy, *Real-Time Anomaly Detection for Streaming Analytics*, 2016, arXiv

[33] Rick Grehan, *Big data showdown: Cassandra vs. HBase*, 2014, InfoWorld *https://www.infoworld.com/article/2610656/database/big-data-showdown–cassandra-vs–hbase.html* (accessed as of January 2018)

[34] Datastax, *Benchmarking top NoSQL Databases*, 2015, End Point

[35] Birendra Kumar Sahu, *A real comparison of NoSQL databases*, 2015 *https://www.linkedin.com/pulse/real-comparison-nosql-databases-hbase-cassandra-mongodb-sahu/* (accessed as of January 2018)

[36] Bogza A.G Adriana-Maria, *Performance evaluation of Apache Mahout for mining large datasets*, Master Thesis, FIB, UPC 2016. Under the supervision of Prof. Fatos Xhafa

[37] DataStax, *Apache Cassandra 3.0 Datastax documentation*, 2018, *https://docs.datastax.com/en/cassandra/3.0/*(accessed as of January 2018)

[38] Andrew Meola, *Internet of Things in Healthcare: Information technology in health*, Business Insider, 2016

[39] Numenta Community, 2018, Introduction to HTM, *https://numenta.org*https://numenta.org (accessed as of January 2018)

[40] Eron Wright, flink-htm GitHub page, 2016 *https://github.com/htm-community/flink-htm*(accessed as of January 2018)

[41] Research and Development Laboratory, Universitat Politécnica de Catalunya, Facultat de Informatica de Barcelona, *https://rdlab.cs.upc.edu/* (accessed as of January 2018)