Real-Time Anomaly Detection from Environmental Data Streams

Chapter · April 2015

DOI: 10.1007/978-3-319-16787-9_8

4 authors, including:

Sven Schade
Geospatial Information Scie…

100 PUBLICATIONS **590** CITATIONS

SEE PROFILE

Joaquín Huerta
Universitat Jaume I

80 PUBLICATIONS **539** CITATIONS

SEE PROFILE

Layout: T1 Standard Unicode | Book ID: 333788_1_En | Book ISBN: 978-3-319-16786-2
Chapter No.: 8 | Date: 23-3-2015 Time: 3:04 pm | Page: 1/20

Real-Time Anomaly Detection from Environmental Data Streams

Sergio Trilles, Sven Schade, Óscar Belmonte and Joaquín Huerta

AQ1

Abstract Modern sensor networks monitor a wide range of phenomena. They are applied in environmental monitoring, health care, optimization of industrial processes, social media, smart city solutions, and many other domains. All in all, they provide a continuously pulse of the almost infinite activities that are happening in the physical space—and in cyber space. The handling of the massive amounts of generated measurements poses a series of (Big Data) challenges. Our work addresses one of these challenges: the detection of anomalies in real-time. In this paper, we propose a generic solution to this problem, and introduce a system that is capable of detecting anomalies, generating notifications, and displaying the recent situation to the user. We apply CUSUM a statistical control algorithm and adopt it so that it can be used inside the Storm framework—a robust and scalable real-time processing framework. We present a proof of concept implementation from the area of environmental monitoring.

AQ2

Keywords Big data and real-time analysis · Environmental sensor data · CUSUM · STORM

S. Trilles (✉) · Ó. Belmonte · J. Huerta
Institute of New Imaging Technologies, Universitat Jaume I, Castellón de la Plana, Spain
e-mail: strilles@uji.es

Ó. Belmonte
e-mail: belfern@uji.es

J. Huerta
e-mail: huerta@uji.es

S. Schade
Institute for Environment and Sustainability, European Commission—Joint Research Centre, Ispra, Italy
e-mail: sven.schade@jrc.ec.europa.eu

F. Bacao et al. (eds.), *AGILE 2015*, Lecture Notes in Geoinformation and Cartography, DOI 10.1007/978-3-319-16787-9_8

Layout: T1 Standard Unicode | Book ID: 333788_1_En | Book ISBN: 978-3-319-16786-2
Chapter No.: 8 | Date: 23-3-2015 | Time: 3:04 pm | Page: 2/20

1 Introduction

Growing amounts of sensor networks measure almost every environmental and man-made phenomena we can think of. These networks can be of different types, but essentially they monitor particular physical characteristics. We can witness some of them in our daily lives, e.g. for environmental monitoring (meteorological, air quality, etc.), health care monitoring, industrial monitoring, or social monitoring/sensing. The Internet of Things (IoT) movement (Kortuem et al. 2010) has allowed these sensor networks to be connected to the Internet, and their access tends to get open to everybody, which ultimately allows to find and retrieve observations in large quantities—and this in every single second.

Each single sensor in each of these networks produces a stream of data and—depending on the particular refresh time—has the capacity to send a large number of measurements. As it becomes difficult to analyze all of these observations in the moment that the raw values are obtained (Manovich 2012), it is challenging to extract relevant knowledge in (near) real-time. This paper introduces this "Big Data" challenge and suggests a mechanism to analyze the arising flood of monitoring data.

Our work particularly addresses anomaly detection. As soon as anomaly is detected, we want to be able to launch a notification in order to (i) trigger a decision-making process, and (ii) inform about the anomaly that caused the event, together with surrounding context information. In many cases, this support has to be provided in real-time because decision-making is time-critical.

We present a system to analyze different sensor networks, to detect anomalies, and to send notifications. We base this system on the Storm framework,[1] which distributively and reliably processes unbounded streams of data in real-time. With this entirely new application of the Storm framework to the IoT, we become able to analyze multiple streams and apply dedicated anomaly-detection algorithms to it. In this pioneering work, we implement the CUmulative SUM (CUSUM) (Page 1954) algorithm in Storm. The resulting system can be applied to any series of values and determine anomalies in real-time. In its first implementation our system operates on environmental sensor networks, where we obtain a series of numerical values. The remainder of the paper is organized as follows. Section 2 shows the Storm framework and enumerates some of its usages. Section 3 presents the CUSUM algorithm and related work. Section 4 introduces the system's design. Section 5 presents a proof of concept for this work. The paper concludes in Sect. 6, which also includes pointers to future work.

[1] Storm, Apache Incubator, http://storm.apache.org.

Layout: T1 Standard Unicode | Book ID: 333788_1_En | Book ISBN: 978-3-319-16786-2
Chapter No.: 8 | Date: 23-3-2015 Time: 3:04 pm | Page: 3/20

2 Storm Framework

Storm is the central component of our proposed solution for processing and detecting anomalies from (big) data streams in real-time. It could be used to execute any algorithms on top of sensor data series. Section 2.1 provides a basic introduction to Storm, while Sect. 2.2 gives the technical background.

2.1 *Introducing Storm*

Storm was created at Backtype, a company that was acquired by Twitter in 2011, became an Apache Incubator project in September 2013, and finally reached the status of an Apache Top-Level Project in September 2014. It is a distributed real-time stream processing framework, which can be used to analyze, filter and normalize data series and apply many others regular-expression filters on data in real-time. The framework is fault tolerant and guarantees data processing.

Storm is comparable to Apache Hadoop (Shvachko et al. 2010). While Hadoop provides programmers a framework for performing batch operations, Storm provides a system for performing streaming computations, similar to Apache S4,[2] Samza,[3] Spark Streaming,[4] and others.

The Storm framework has already been used for a wide range of purposes:

- Simoncelli et al. (2013) used Storm to extract trending hashtags from Twitter, and count occurrences of citations.
- Sunderrajan et al. (2014) used it to detect outliers in the power consumption from smart plugs.
- Yutan et al. (2014) used Storm to monitor the flow of data and locate the source of attacks or anomalies in real-time by finding the specific IP addresses.
- Storm has also been used to apply data mining in real-time sources (De Francisci Morales 2013).
- Sitaram et al. (2013) applied Storm for speech recognition in real-time.

We suggest to use Storm in the context of the IoT, and especially for environmental monitoring, because it allows the finest-grained control of processing unit, offers greatest control over data streams, is mature, and has a strong community.

[2] Apache S4, Apache Incubator, http://incubator.apache.org/s4.
[3] Samza, Apache Incubator, http://samza.incubator.apache.org.
[4] Spark Streaming, Apache Incubator, http://spark.apache.org/streaming.

Layout: T1 Standard Unicode | Book ID: 333788_1_En | Book ISBN: 978-3-319-16786-2
Chapter No.: 8 | Date: 23-3-2015 Time: 3:04 pm | Page: 4/20

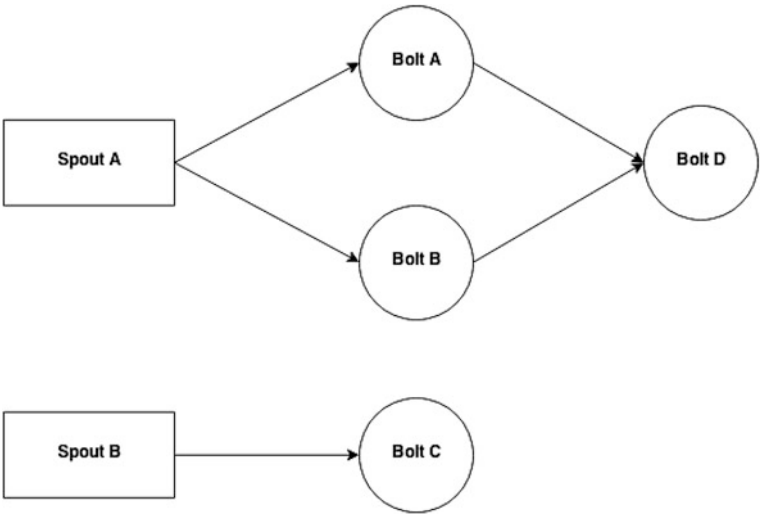

Fig. 1 An example of Storm *Topology* with different *Spouts* and *Bolts*

2.2 Technical Background of Storm

Storm has its own terminology, starting with the concept of *Topologies*. A *Topology* defines the way (workflow) in which a problem will be processed. It is similar to a job in Hadoop. However, Hadoop jobs will have an end, while the *Topology* will always run because there is a need for continuous computation.

Figure 1 illustrates a *Topology* as the workflow of *Spouts* and *Bolts*. The *Spouts* handle the insertion of data *tuples* into the *Topology* and send each of these *tuples* to *Bolts*. They thus connect the input stream and send the values to the *Bolt* that is responsible for the stream processing. Each *Bolt* processes the streams that it receives from the *Spout*. It applies any particular procedure to generate its output stream. The actual improvement of Storm compared to other solutions, is that these operations can be parallelized. Parallelisation is handled on the level of a single *Bolt* and the parallelization factor is defined as part of the *Topology*.

Storm offers an at-least-once processing guarantee, but does not consider the order in which data streams are emitted. In fact, the *tuples* will have a different order when they are processed. For the objective of this work is necessary to maintain the order of the *tuples*. We use Trident[5] to ensure exactly this. Trident is a high-level abstraction framework for computation on Storm. As with the core Storm Application Programming Interface (API), Trident uses *Spouts* as the source of data streams. It has consistent, exactly-once semantics (same order), so it is easy to

[5] Storm Trident, Apache Incubator, http://storm.apache.org/documentation/Trident-API-Overview.html.

Layout: T1 Standard Unicode | Book ID: 333788_1_En | Book ISBN: 978-3-319-16786-2
Chapter No.: 8 | Date: 23-3-2015 Time: 3:04 pm | Page: 5/20

reason about Trident *Topologies*. Trident already offers different operations, such as functions, filters and create aggregations from streams of *tuples*.

3 CUmulative SUM (CUSUM)

We selected the CUSUM algorithm for detecting anomalies in data series from environmental monitoring. In essence, this algorithm compares two different instances of the same discrete probability function for a data series (Mesnil and Petitgas 2009). The algorithm was initially developed for industrial control purposes. In recent years, it has been successfully used in other areas. An overview of past applications is provided in Sect. 3.1. The details of the CUSUM algorithm are depicted in Sect. 3.2.

3.1 Applications of CUSUM

Osanaiye and Talabi (1989), for instance, used the CUSUM algorithm in order to detect possible outbreaks of epidemics. Grigg et al. (2003) analyzed the 30-day mortality for patients after cardiac surgery. Furthermore, CUSUM used is to improve the communication in Wireless Sensor Networks. Jeske et al. (2009) developed two CUSUM change-point detection algorithms for data network monitoring applications. CUSUM has additionally been used in pattern recognition, specifically in neural networks. Sample of them is Guh and Hsieh (1999) study where it proposes an artificial neural network based model, which contains several back propagation networks. Chuen-Sheng (1995) described an alternative approach for statistical process control, using artificial neural network technology and compares its performance with that of the combined Shewhart-CUSUM schemes.

More recently, CUSUM has been adapted and used for a number of environmental problems, including the following works:

- Barratt et al. (2007) used the CUSUM algorithm to detect anomalies in carbon monoxide (CO) levels after the introduction of a permanent bus line in Marylebone Road, London.
- Carslaw et al. (2006) presented an analysis of the same data source as the Barratt et al. (2007) study. They analyze various components with CUSUM, such as nitrogen oxides (NOx), nitrogen dioxide (NO_2), particulate matter with diameter less than 10 μm (PM10), particulate matter with diameter less than 25 μm PM25, and PMcoarse (defined as PM25-10).
- Chelani (2011) compares a modified CUSUM algorithm with the original to detect anomalies in phenomena, such as NO_2, CO and PM10. The paper concludes that the modified CUSUM can help in detecting anomalies when there is greater variability in the observations.

Layout: T1 Standard Unicode | Book ID: 333788_1_En | Book ISBN: 978-3-319-16786-2
Chapter No.: 8 | Date: 23-3-2015 | Time: 3:04 pm | Page: 6/20

- Charles and Jeh-Nan (2002) proposed to use the CUSUM control chart to monitor data about industry emissions to the environment to detect abnormal changes in a timely manner.

Following these success stories, we decided to use CUSUM as part of our anomaly detection system. In our proof of concept implementation (Sect. 5) CUSUM is applied to anomaly detection of air pollutants.

3.2 Technical Background of CUSUM

CUSUM considers a set of observations (x_i) with collected observation $i = 1, \ldots, n$, where n is the number of data points. The algorithm assumes that these observations are in-control when the collection has a mean (μ) and standard deviation (σ^2) for a normal period and following a normal distribution $N(\mu, \sigma^2)$. When the process is in-control, we can obtain the CUmulative SUM (S_i) in an iterative way through the following expression:

$$S_i = S_{i-1} z_i \tag{1}$$

where $S_0 = 0$, z_i is the standard normal variable, $z_i = \frac{x_i - \bar{x}}{s}$, $\bar{x}$ is the mean and s is the standard deviation of the time series. Further, the change in terms of increased or decreased process mean can be detected, respectively by computing the quantities as (Lucas 1982):

$$\begin{aligned} S_{H_i} &= MAX[0, (z_i - k) + S_{H_i - 1}] \\ S_{L_i} &= MIN[0, (z_i + k) + S_{L_i - 1}] \end{aligned} \tag{2}$$

where the parameter k is the reference value to be chosen appropriately. The parameter, k, is the allowable "slack" in the process and is usually set to be one half of the mean one wishes to detect. The confidence limits (*threshold*) specified for the CUSUM control charts are $\hat{A} \pm h\sigma_x$, where $h = 5$ and σ_x is the standard deviation (Barratt et al. 2007).

When S_{H_i} or S_{L_i} overcome the *threshold*, the algorithm detects an anomalies. If S_{H_i} exceeds the *threshold* the anomaly will be due to the increase (*up-event*). And If S_{L_i} is greater than the *threshold*, it will be due to the decrease (*down-event*).

Two characteristics of CUSUM limit the sensitivity of results. First, the identification of an out-of-control process relies on the assumption that readings are statistically independent and follow a normal distribution. Second, phenomenon measurements can have some seasonality and long-term trends. This has the effect that the *threshold* may be out of adjustment.

Layout: T1 Standard Unicode | Book ID: 333788_1_En | Book ISBN: 978-3-319-16786-2
Chapter No.: 8 | Date: 23-3-2015 Time: 3:04 pm | Page: 7/20

4 System Design

This section presents our system design for real-time anomaly detection. This system is formed by four components: data streams from sensor networks; the Real-time Message Service (RMS) (that equally handles the input as well as the output data flows); *Topology* developed in Storm, including our implementation of the CUSUM algorithm for real-time anomaly detection; and the application to visualize the final outcomes. Figure 2 shows these components and the connections between them. The follow subsections detail the central components.

4.1 Real-Time Message Service

We have to handle observations in real time, i.e. as soon as new values become available on the Web. Traditionally, web-based resources are accessed by client's HTTP requests to a server, which then responds by returning the resource. This procedure should be repeated every time that the user wants to access the resource. For data sources from high refresh rates, such as stock price or environmental sensor data, require frequent requests to the server in order to (almost) constantly receive the latest data sets.

More effective and efficient approaches have been developed to address such cases. They are based on polling mechanisms, where the client repeatedly sends new requests to the server. If the server has no new data, then it sends appropriate indication and closes the connection. The client then waits a bit and sends another request after some time.

A more recent approach is long-polling. In this case, the client sends a request to the server and the server keeps the connection open for a previously defined period, or a timeout period. When the server has new data available, then it directly transmits it to the client. An example of this approach is the Message Oriented

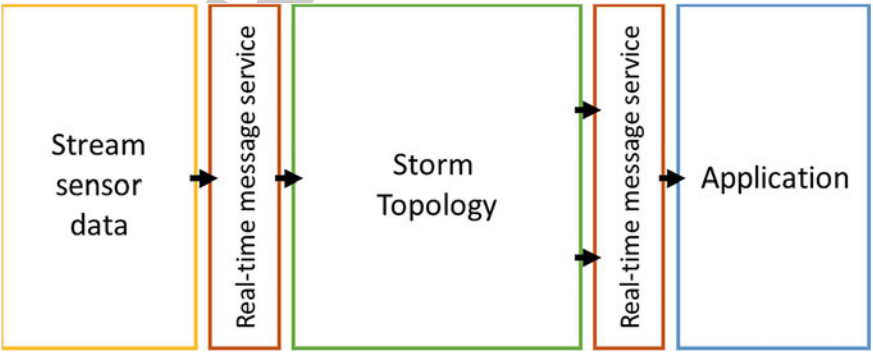

Fig. 2 System design for the anomalies detection

Layout: T1 Standard Unicode | Book ID: 333788_1_En | Book ISBN: 978-3-319-16786-2
Chapter No.: 8 | Date: 23-3-2015 Time: 3:04 pm | Page: 8/20

Middleware, which provides an infrastructure to send and receive messages between distributed systems.

We decided to use this approach in order to realize the real-time communication in our system. This component is called RMS. It is used to serve the data provided by the sensors and is based in Java Message Service (JMS) (Hapner et al. 2002) to deliver asynchronous communication via the web. JMS is a standard for the implementation of Message Oriented Middleware on the Java platform for sending messages between two or more parts. In the JMS context, exists different components:

- *Provider*: an implementation of the JMS interface for a Message Oriented Middleware.
- *Client*: an application or process that produces and/or receives *Messages*.
- *Producer/Publisher*: a *Client* that creates and sends *Messages*.
- *Consumer/Subscriber*: a *Client* that receives *Messages*.
- *Message*: a *Message* can be any object or data that needs to be transported using JMS.
- *Queue*: a staging area that contains *Messages* that have been sent and are waiting to be read (by only one *Consumer*). A *Queue* only guarantees that each *Message* is processed only once.
- *Topic*: a distribution mechanism for publishing *Messages* that are delivered to multiple *Subscribers*.

JMS has two different communication models, which are both relevant for our work (see also Fig. 3):

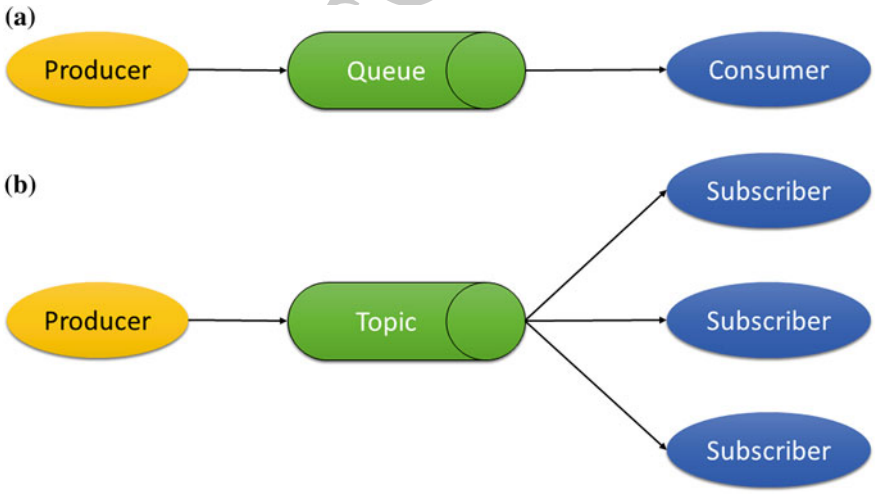

Fig. 3 JMS communication models. **a** *Point-to-point* and **b** *Publish/Subscribe*

Layout: T1 Standard Unicode | Book ID: 333788_1_En | Book ISBN: 978-3-319-16786-2
Chapter No.: 8 | Date: 23-3-2015 | Time: 3:04 pm | Page: 9/20

- *Point-to-point* model: in this model, the *Messages* are routed to an individual *Consumer* which maintains a *Queue* of "incoming" *Messages* (Fig. 3a).
- *Publish/subscribe* model: this model supports publishing *Messages* to a particular *Message Topic*. *Subscribers* may register interest in receiving *Messages* on a particular *Message Topic* (Fig. 3b).

For the development of the RMS, we have used a JMS framework called ActiveMQ.[6] ActiveMQ is a popular and powerful open source persistence messaging and integration patterns server with scheduler capabilities, acts as a message broker in the framework. It support different protocols, such as: OpenWire,[7] REST (Fielding 2000), Simple (or Streaming) Text-Oriented Messaging Protocol (STOMP),[8] Web Socket (Hickson 2011) and more. For our work we used a STOMP interface. It is a simple text-oriented protocol, similar to HTTP. STOMP provides an interoperable wire format that allows clients to communicate with almost every available message broker.

We provide a *Client*, called JMS *Client*, to connect to RMS via the STOMP interface. It offers the different models of connections (both *Point-to-point* model as *Publish/subscribe*). This *Client* can be to perform as *Producer* or *Consumer*. It will be used into the stream sensor data, the Storm *Topology* and the application.

4.2 Stream Sensor Data

We build on our previous activities (Trilles et al. 2014) in order to access sensor data sources. In that work, wrapping techniques were applied to obtain sensor observations from each individual sensor. For the work at hand, we re-use these techniques, but now serve the observations via real-time interfaces inside the RMS. Each time when a new observation is produced by a sensor, it becomes directly distributed by the RMS via its real-time interface. At this occasion, we apply the *Point-to-point* model of JMS, because it ensures that the receiver will get all the produced *Messages*.

In order to connect with the RMS, the JMS *Client* (introduced in the Sect. 4.1) has been used as a *Producer*. Any new observation is sent to the corresponding *Queue*. One *Queue* per phenomenon and sensor. The Fig. 4 illustrates this use of the RMS.

To indicate the correct parameters to connect to each *Queue*, an eXtensible Markup Language (XML) file has been created. It contains a single entry to define the sensor network, contains details about each sensor: an identifier, name, and city, state or location. The sensor entry includes a separate element for each phenomenon that is measured by the sensor. Each of these elements contains details about the

[6] ActiveMQ framework, Apache Software Foundation, http://activemq.apache.org.
[7] OpenWire protocol, Apache Incubator, http://activemq.apache.org/openwire.html.
[8] STOMP protocol, Apache Incubator, http://activemq.apache.org/stomp.html.

Layout: T1 Standard Unicode | Book ID: 333788_1_En | Book ISBN: 978-3-319-16786-2
Chapter No.: 8 | Date: 23-3-2015 Time: 3:04 pm | Page: 10/20

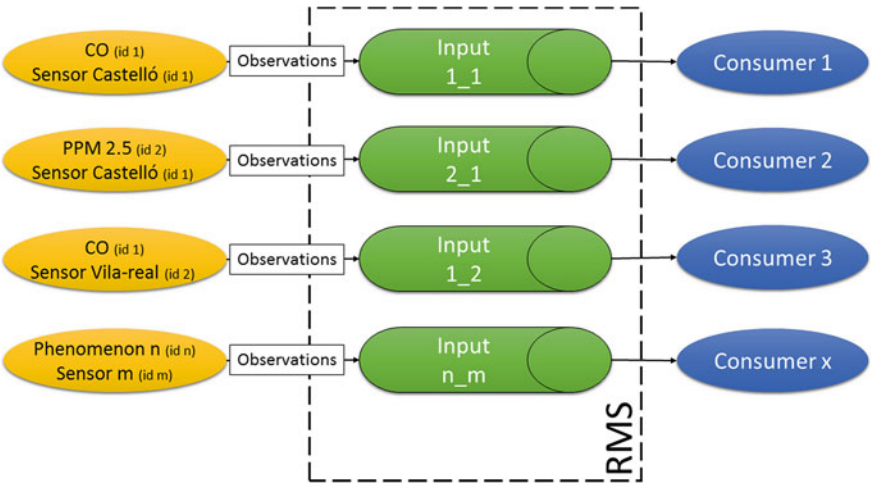

Fig. 4 The figure shows how RMS is used for the stream sensor data

measured phenomenon: an identifier, observed property, unit of measure. It also contains the parameters that are needed to run CUSUM (threshold and k).

A proprietary format has been used for encoding each observation. This observation contains an identifier, the measured value, the time-stamp and the sensor identifier. It is encoded using JavaScript Object Notation (JSON).

4.3 Storm Topology

Now that the real-time interfaces are available to serve a sensor data, we can concentrate on the *Topology* for anomaly detection. Within the *Topology* definition, we call the sensor observations *tuples*. The *Bolts* are responsible for processing these *tuples*. In our case, the *Topology* consists of one *Spout* and two separated *Bolts* (Fig. 5). We visit each of them in detail below.

The *Spout* is called *Real-time Spout* and aims to connect with the RMS to obtain the *tuples*. In this *Spout* the JMS *Client* has been used as *Consumer*. As soon as the RMS provides a new observation of the stream, it is read by the *Real-time Spout*. In the *Point-to-point* model, when the *Spout* reads a *tuple*, it is deleted from the *Queue*. This *Spout* is responsible for transforming the *Message* and creating the *tuples* that will be passed to the *Bolts*. It also uses the information from the XML files to connect to the RMS.

The first *Bolt*, called *Observations Bolt*, is only responsible for providing the latest *tuples* that the *Spout* has sent. The *Observation Bolt* is necessary to serve the *tuples* in a uniform way and these can be consumed by the final applications. It has two different functionalities. First, when the *Bolt* receives a new *tuple* from the

Layout: T1 Standard Unicode | Book ID: 333788_1_En | Book ISBN: 978-3-319-16786-2
Chapter No.: 8 | Date: 23-3-2015 Time: 3:04 pm | Page: 11/20

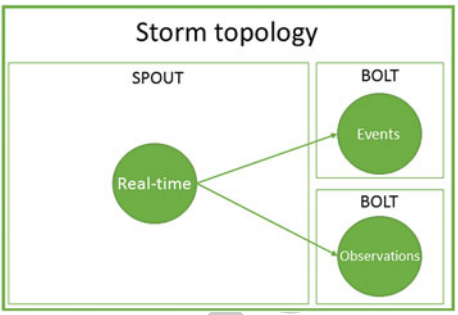

Fig. 5 The *Topology* created to apply CUSUM in the sensor data

Real-time Spout, it sends the observation using the RMS. Inside the *Bolt*, the JMS *Client* has been used as *Producer* to connect with the RMS. The *Observations Bolt* creates different *Queues* per phenomenon and sensor. Secondly, this *Bolt* serves a set of the latest *tuples* that were sent by the *Real-time Spout*. A First-In, First-Out (FIFO) buffer is used to store a few previous *tuples*—one for each phenomenon that the system supports. When receives a new *tuples*, the *Observations Bolt* adds the *tuple* to the buffer and the oldest *tuple* is removed. In this way, the user can access a small range of previous observations and does not see only the last observation. These observations are also offered using the RMS. The functionalities are both realised with the *Publish/Subscribe* model, so that different *Consumers* (final applications) can connect to the same *Queue*.

The *Events Bolt* is responsible to apply CUSUM to the series of *tuples* that are provided by the *Spout*. As already mentioned, Storm only ensures that all *tuples* are processed, but it does not guarantee the processing order of the *tuples* will be the same with that have read. Trident has been used to solve this inconvenience.

In order to execute the CUSUM algorithm on each phenomenon-specific measurement stream (as described in Sect. 3.2), we separate the tuples, which arrive from the sensor network, using unique phenomenon identifiers. The pseudocode below resumes the real-time variation for CUSUM. For the execution, we facilitate a data structure that cumulates and stores the sum for each iteration. Finally, we evaluate if the recent *tuple* causes an event by using a dedicated threshold.

When an anomaly is detected by CUSUM, the *Events Bolt* uses the JMS *Client* to send notifications to the RMS. Again, a *Queue* is created for each phenomenon and sensor. We again apply the *Publish/Subscribe* model (Fig. 6).

The events are currently also provided in a proprietary format with JSON encoding. Each event contains a sensor identifier and the identifier of the particular observation that has caused the event, together with a string value that indicates the identified event refers to an exceedance of the threshold (up-event) or to the fact that the observed value falls below the thresholds (down-event).

Layout: T1 Standard Unicode | Book ID: 333788_1_En | Book ISBN: 978-3-319-16786-2
Chapter No.: 8 | Date: 23-3-2015 Time: 3:04 pm | Page: 12/20

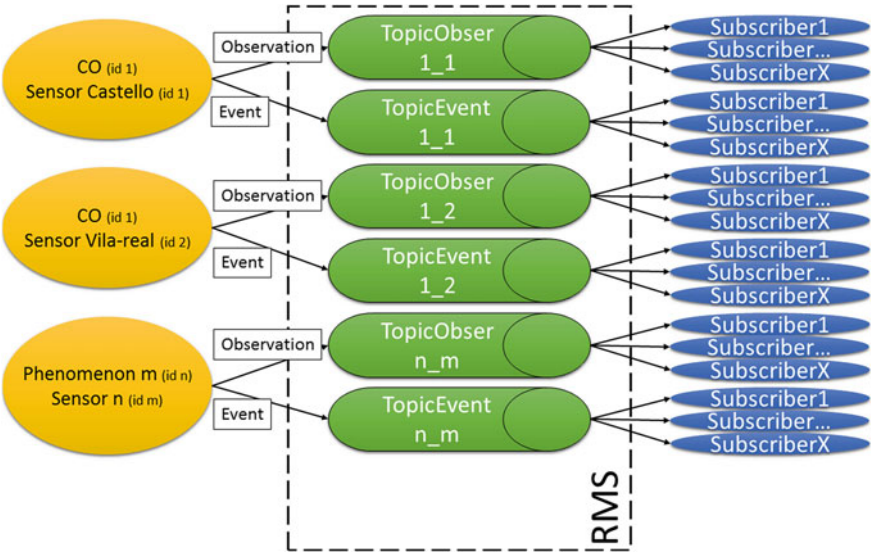

Fig. 6 The figure shows how is used RMS for the outputs, both for observations and events

4.4 Application Able to Connect to the RMS

An application has been designed in order to visualize the RMS outputs. The application has a single main use case (*Sensor map*) to visualize the real-time observations and the events (Fig. 7). This use case shows the sensors provided from the XML file through the *Sensor data sensor settings*.

To obtain the RMS outputs, the *Sensor map* exclude two sub-use cases, *Sensor observation* and *Sensor event*. The former is responsible to connect to the RMS and

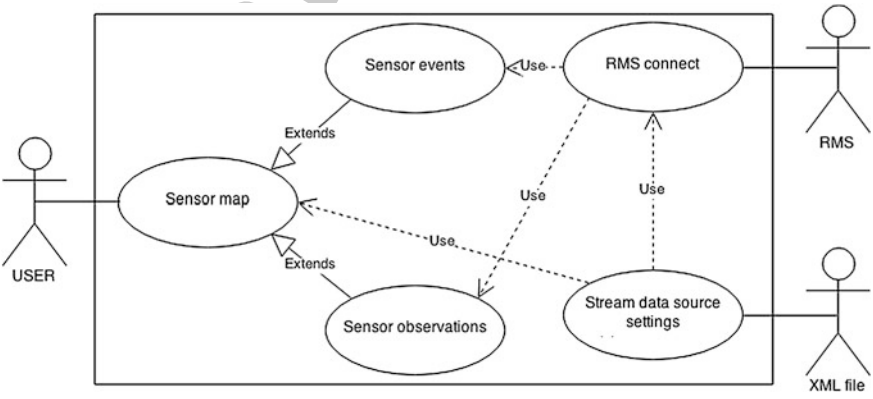

Fig. 7 The use case diagram for the designed application

Layout: T1 Standard Unicode | Book ID: 333788_1_En | Book ISBN: 978-3-319-16786-2
Chapter No.: 8 | Date: 23-3-2015 Time: 3:04 pm | Page: 13/20

retrieve the historical and last observations. The latter connects to the RMS for obtaining. The *Sensor event* connects to the RMS for obtaining the generated events. Both use cases apply the *RMS connect* as a *Subscriber* JMS *Client* in order to establish the connection with the RMS and obtain the *Topics* from the different subscribed *Queues*. This use case obtains the settings, which are required to connect with the RMS, from the *Sensor data sensor settings*.

Algorithm 1 CUSUM algorithm for real-time

procedure CUSUM REAL-TIME
  $numPhen =$ num of phenomena in the system
  $S_{previous_{high}}[numPhen] = 0$
  $S_{previous_{low}}[numPhen] = 0$
  while new observation $! = false$ **do**
    $observation =$ value of the new observation
    $idPhen =$ identifier of the phenomenoun
    $SNV = \frac{observation - \mu[idPhen]}{\sigma[idPhen]}$
    $S_{current_{high}}[idPhen] = MAX[0, (SNV - k[idPhen]) + S_{previous_{high}}[idPhen]]$
    $S_{current_{low}}[idPhen] = MIN[0, (SNV + k[idPhen]) + S_{previous_{low}}[idPhen]]$
    if $S_{current_{high}}[idPhen] >= threshold[idPhen]$ **then**
      Send "Up-Event"
      $S_{current_{high}}[idPhen] = 0$
      $S_{current_{low}}[idPhen] = 0$
    end if
    if $S_{current_{low}}[idPhen] >= threshold[idPhen]$ **then**
      Send "Down-Event"
      $S_{current_{low}}[idPhen] = 0$
      $S_{current_{high}}[idPhen] = 0$
    end if
    $S_{previous_{high}}[idPhen] = S_{current_{high}}[idPhen]$
    $S_{previous_{low}}[idPhen] = S_{current_{low}}[idPhen]$
  end while
end procedure

5 Proof of Concept

We implemented a proof of concept for the proposed system. The air quality network of the Valencian Community government[9] is used for testing purposes and an event dashboard illustrates the outcomes. Figure 8 shows all involved components.

[9] http://www.citma.gva.es/web/calidad-ambiental/datos-on-line.

Layout: T1 Standard Unicode | Book ID: 333788_1_En | Book ISBN: 978-3-319-16786-2
Chapter No.: 8 | Date: 23-3-2015 Time: 3:04 pm | Page: 14/20

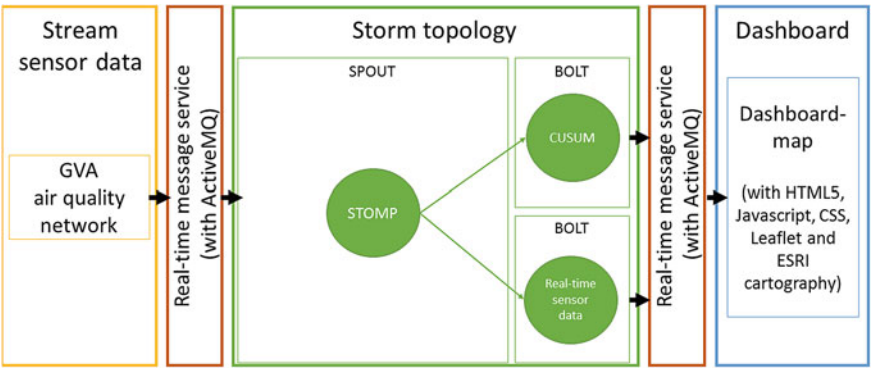

Fig. 8 The system design details for the proof of concept

5.1 Example Dataset

The Valencia Community government has deployed a network of 61 active stations for measuring the air quality in the region (Fig. 9). Its stations measure levels of multiple pollutants and meteorological conditions in urban, rural and industrial areas. These pollutants include sulphur dioxide (SO_2), nitrogen monoxide (NO), nitrogen dioxide (NO_2), nitrogen oxides (NOx), carbon monoxide (CO), ozone (O_3), benzene (C_6H_6) and other hydrocarbons such as toluene and xylene; particulate matter with diameter less than 10 μm (PM10), 2.5 μm (PM2.5), and 1 μm (PM1). The sensor network also covers metal levels such as arsenic, nickel, cadmium, lead and polycyclic aromatic hydrocarbons on the PM10 fraction. Some stations are also equipped with meteorological sensors for measuring parameters, such as wind speed and direction, relative humidity, solar radiation, atmospheric pressure and precipitation. The measurements are published on a website that is updated hourly. In this way, this data source provides both: historical and real-time data, which we need to test the proposed method.

As earlier commented, to use the CUSUM algorithm we need two parameters (threshold and k) per phenomenon that our system analyses. We obtain these parameters with historical data from the presented data source. For this, we have used one year of historical data (1st January 2013 to 31st December 2013). To apply CUSUM, and following the work of others (see Sect. 3.1), we are considering that all analyzed phenomena follow a normal distribution.

5.2 Event Dashboard

We developed an event dashboard to present the data provided by the RMS. All sensors of a network can be displayed on a map using markers (Fig. 10a). Inside the

Layout: T1 Standard Unicode | Book ID: 333788_1_En | Book ISBN: 978-3-319-16786-2
Chapter No.: 8 | Date: 23-3-2015 Time: 3:04 pm | Page: 15/20

Fig. 9 Map of the 61 air quality stations in the Valencia network AQ3

marker appears the amount of events that have been caused by this particular sensor. If this sensor triggers an event it appears in red. The dashboard does not differ whether the event is "up" (exceedance of the threshold) or "down" (falling under the threshold). A scale clustering has been applied to the markers following the quantity of events (Fig. 10b). When zooming out, the markers will be combined to the total amount of events that have been launched inside the cluster. The colour of the marker will be red if one sensor of this cluster launched an event.

When a user selects a single sensor marker, new markers appear as a menu (Fig. 11a). Each new marker symbolizes a phenomenon that is associated to this particular sensor, the names of the phenomena and the marker will appear in red, if a new event has been reported. Upon mouse click on one of the phenomena marker, the dashboard displays a pop-up widget that displays the latest observations in a graph (Fig. 11b). These observations are obtained from the buffer of the RMS. The graph is dynamically updated with the latest observations from the RMS. In the graph, events are highlighted as red rhombus. Also, these events are obtained from the RMS. The chart can display each of the observations interactively. The user can display different graphs simultaneously, even different phenomena from different sensors. In this way, one can compare the values of for same phenomena inside the same network.

Layout: T1 Standard Unicode | Book ID: 333788_1_En | Book ISBN: 978-3-319-16786-2
Chapter No.: 8 | Date: 23-3-2015 Time: 3:04 pm | Page: 16/20

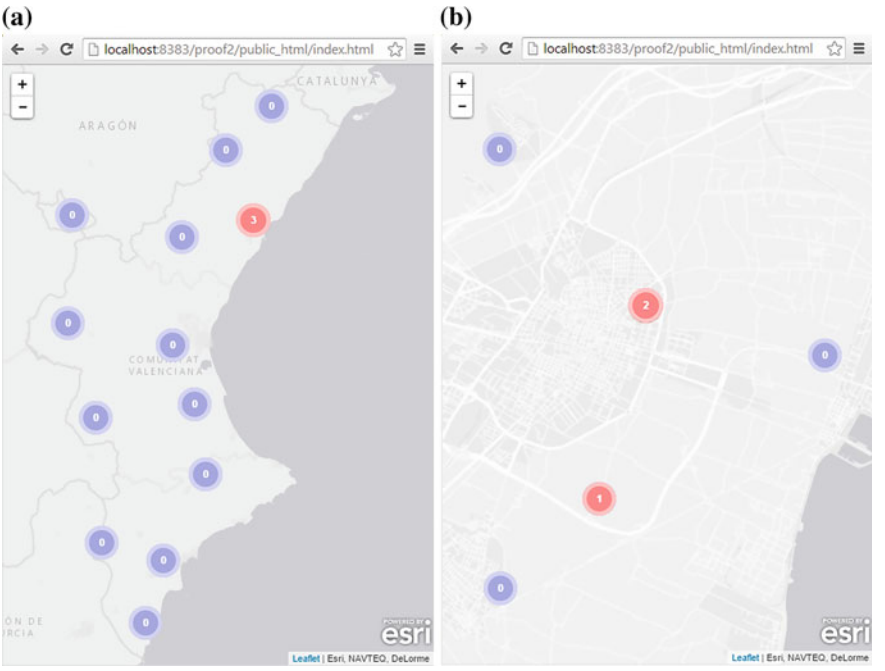

Fig. 10 **a** It shows the sensors (or cluster) as markers and the events are indicated inside the marker. **b** The figure compared with (**a**), shown as applied clustering by the amount of launched events

In order to implement a flexible, compatible and standards-compliant solution, we re-used a combination of already existing frameworks:

- Leaflet[10] with ESRI cartography,[11] to put the makers on the map. It proved to be fast and efficient. In addition, it can execute in a restrictive environments, such as smartphones.
- Another library that we used is Bootstrap.[12] It offer the capacity to building a responsive dashboard, can adapt to the device features. Also, we use jQuery [13] to handle popups.
- Finally other framework used is Highchart JS.[14] It is a graphics library written in HTML5 and JavaScript. The library provides an easy and interactive way to generate graphs in a web environment.

[10]Leaflet: An Open-Source JavaScript Library for Mobile-Friendly Interactive Maps, http://leafletjs.com.
[11]ESRI, http://www.esri.com.
[12]Bootstrap, Twitter http://getbootstrap.com.
[13]jQuery, jQuery Foundation http://jquery.com.
[14]Highcharts JS, Highcharts AS http://www.highcharts.com.

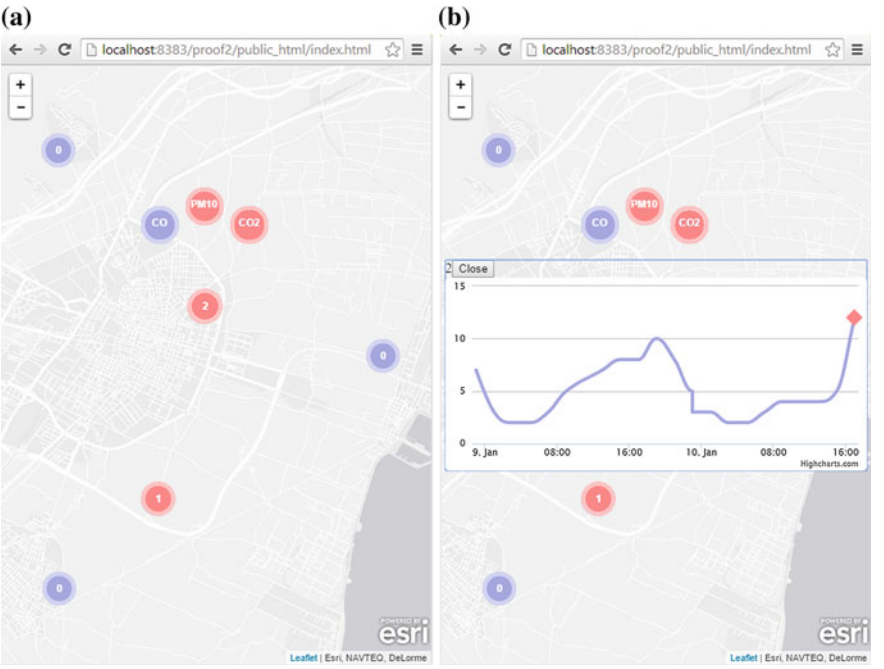

Fig. 11 **a** It shows A menu to visualize phenomena is show in each marker. **b** A pop-up with a graph are displayed to visualize the last observations and the events launched

6 Conclusions

The ever-increasing amount of sensors challenges the real-time detection of anomalies (Morris and Paradiso 2002). We propose a methodology and a system to analyze data streams from sensor networks near to real-time. This paper presented our system design in which we use novel technologies, such as the stream processing framework Storm, and Java Message Service (JMS). To our knowledge this is the first time that Storm is applied in the IoT and environmental monitoring.

In order to detect events in the streams of observations, we have developed a variation of CUSUM algorithm (Page 1954) so that it can run inside the Storm framework. Apart from the overall anomaly-detection solution, we also developed a dashboard application to visualize events that have been detected by the system, together with the contextual sensor data. A proof of concept implementation illustrated the overall feasibility of the approach using an example data source from environmental monitoring. CUSUM has strict underlying assumptions on value distributions, but it provides good results when it is applied on this type of data (Barratt et al. 2007; Barratt and Fuller 2014).

The primary challenge of the presented work was to provide a system able to analyze data from sensors and detect abrupt changes in a near the real-time.

Layout: T1 Standard Unicode | Book ID: 333788_1_En | Book ISBN: 978-3-319-16786-2
Chapter No.: 8 | Date: 23-3-2015 Time: 3:04 pm | Page: 18/20

Our system allows to analyze each of the observations without faults. Although, the used data source has low refresh rates, it could scale to higher refresh rates or add multiple data sources into the same system. As Storm was already successfully used in various other application areas, we are confident that integrations can be realised on top of this framework.

The second challenge was to detect abrupt changes in observation series. CUSUM has proven to be useful in looking at anomalies in the series of observations of air quality and weather. Although, we have validated our proposal with a particular data source, our overall system is designed to work simultaneously with multiple data sources of diverse characteristics, either with different kinds of interface connection or different data encodings. However, CUSUM presents some limitations that must be taken into account, such as the consideration of all the series follow a normal distribution and a series of observations cannot have trend changes. The use of alternatives algorithms, for example the one developed by Chelani (2011), remains to be investigated. This should particularly consider those algorithms that can account for phenomenon-specific probability distributions.

Now that the overall work-flow has been put into place, we concentrate on the standards support of the involved components. In the next step, we migrate the exchange formats to the Observations and Measurements standard (O&M) (Cox 2007) of the Open Geospatial Consortium (OGC). This is most important for encoding the output of the event detection components in a way that it can be easily integrated into diverse application tools, and to ensure interoperability with third-party systems.

Considering the input that is coming from any kind of sensor that produces values in metric scales, we will also soon support O&M as a format. In the medium term, we plan to extend the input related part of the system beyond O&M. We plan to apply a brokering solution (Buschmann et al. 1996; Nativi et al. 2012), so that inputs of multiple types can be accepted. In this way, the overall system will be able to operate on the most common standards for encoding measurements, but remains flexible for adoption new (even proprietary) formats.

We also plan to test the scalability of the system. This will be using a larger number of sources and a higher refresh rates. These experiments should have not affect on performance because Storm has been explicitly designed for scalability in order to be able to deal with huge amounts of data.

Acknowledgments This work has been supported in part by European Commission and Generalitat Valenciana government (grants ACIF/2012/112 and BEFPI/2014/067).

References

Barratt, B., & Fuller, G. (2014). Intervention assessments in the control of PM10 emissions from an urban waste transfer station. *Environmental Science: Processes and Impacts, 16*(6), 1328–1337.

Barratt, B., Atkinson, R., Anderson, H., Beevers, S., Kelly, F., Mudway, I., & Wilkinson, P. (2007). Investigation into the use of the cusum technique in identifying changes in mean air pollution levels following introduction of a traffic management scheme. *Atmospheric Environment, 41*(8), 1784–1791.

Layout: T1 Standard Unicode | Book ID: 333788_1_En | Book ISBN: 978-3-319-16786-2
Chapter No.: 8 | Date: 23-3-2015 Time: 3:04 pm | Page: 19/20

Buschmann, F., Meunier, R., Rohnert, H., & Sommerlad, P. (1996). *Pattern-oriented software architecture: A system of patterns* (Vol. 1). New York: Wiley.

Carslaw, D., Ropkins, K., & Bell, M. (2006). Change-point detection of gaseous and particulate traffic-related pollutants at a roadside location. *Environmental Science and Technology, 40*(22), 6912–6918.

Charles, J., & Jeh-Nan, P. (2002). Evaluating environmental performance using statistical process control techniques. *European Journal of Operational Research, 139*(1), 68–83.

Chelani, A. (2011). Change detection using cusum and modified cusum method in air pollutant concentrations at traffic site in Delhi. *Stochastic Environmental Research and Risk Assessment, 25*(6), 827–834.

Chuen-Sheng, C. (1995). A multi-layer neural network model for detecting changes in the process mean. *Computers and Industrial Engineering, 28*(1), 51–61.

Cox, S. (2007). Observations and measurements part 1—observation schema. Technical report, Open Geospatial Consortium (OGC).

De Francisci Morales, G. (2013). Samoa: A platform for mining big data streams. In *Proceedings of the 22nd International Conference on World Wide Web Companion, International World Wide Web Conferences Steering Committee, Republic and Canton of Geneva, Switzerland, WWW '13 Companion* (pp. 777–778).

Fielding, R. (2000). Representational state transfer (rest) (Chap. 5). Fielding Dissertation.

Grigg, O., Farewell, V., & Spiegelhalter, D. (2003). Use of risk-adjusted CUSUM and RSPRTcharts for monitoring in medical contexts. *Statistical Methods in Medical Research, 12*(2), 147–170.

Guh, R., & Hsieh, Y. (1999). A neural network based model for abnormal pattern recognition of control charts. *Computers and Industrial Engineering, 36*(1), 97–108.

Hapner, M., Burridge, R., Sharma, R., Fialli, J., & Stout, K. (2002). *Java message service*. Santa Clara, CA: Sun Microsystems Inc.

Hickson, I. (2011). The WebSocket API. W3C Working Draft WD-websockets-20110929, September.

Jeske, D., Montes De Oca, V., Bischoff, W., & Marvasti, M. (2009). Cusum techniques for timeslot sequences with applications to network surveillance. *Computational Statistics Data Analysis, 53*(12), 4332–4344.

Kortuem, G., Kawsar, F., Fitton, D., & Sundramoorthy, V. (2010). Smart objects as building blocks for the internet of things. *IEEE Internet Computing, 14*(1), 44–51.

Lucas, J. (1982). Combined Shewhart-CUSUM quality control schemes. *Journal of Quality Technology, 14*(2).

Manovich, L. (2012). Trending: The promises and the challenges of big social data. In M. K. Gold (Ed.), *Debates in the digital humanities* (pp. 460–475). Minneapolis: U of Minnesota P.

Mesnil, B., & Petitgas, P. (2009). Detection of changes in time-series of indicators using cusum control charts. *Aquatic Living Resources, 22*(02), 187–192.

Morris, S., & Paradiso, J. (2002). Shoe-integrated sensor system for wireless gait analysis and real-time feedback. In *Engineering in Medicine and Biology, 2002. 24th Annual Conference and the Annual Fall Meeting of the Biomedical Engineering Society EMBS/BMES Conference, 2002. Proceedings of the Second Joint, IEEE* (Vol. 3, pp. 2468–2469).

Nativi, S., Craglia, M., & Pearlman, J. (2012). The brokering approach for multidisciplinary interoperability: A position paper. *International Journal of Spatial Data Infrastructures Research, 7*, 1–15.

Osanaiye, P., & Talabi, C. (1989). On some non-manufacturing applications of counted data cumulative sum (CUSUM) control chart schemes. *The Statistician, 38*, 251–257.

Page, E. (1954). Continuous inspection schemes. *Biometrika, 41*(1/2), 100–115.

Shvachko, K., Kuang, H., Radia, S., & Chansler, R. (2010). The Hadoop distributed file system. In *2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST), IEEE* (pp. 1–10).

AQ4

Layout: T1 Standard Unicode | Book ID: 333788_1_En | Book ISBN: 978-3-319-16786-2
Chapter No.: 8 | Date: 23-3-2015 Time: 3:04 pm | Page: 20/20

Simoncelli, D., Dusi, M., Gringoli, R., & Niccolini, S. (2013). Stream-monitoring with blockmon: Convergence of network measurements and data analytics platforms. *SIGCOMM Computer Communication Review, 43*(2), 29–36.

Sitaram, D., Srinivasaraghavan, H., Agarwal, K., Agrawal, A., Joshi, N., & Ray, D. (2013). Pipelining acoustic model training for speech recognition using storm. In *2013 Fifth International Conference on Computational Intelligence, Modelling and Simulation (CIMSim)* (pp. 219–224).

Sunderrajan, A., Aydt, H., & Knoll, A. (2014). Real time load prediction and outliers detection using storm. In *Proceedings of the 8th ACM International Conference on Distributed Event-Based Systems, ACM, New York, NY, USA, DEBS '14* (pp. 294–297).

Trilles, S., Belmonte, O., Diaz, L., & Huerta, J. (2014). Mobile access to sensor networks by using GIS standards and restful services. *IEEE Sensors Journal, 14*(12), 4143–4153.

Yutan, D., Jun, L., Fang, L., & Luying, C. (2014). A real-time anomalies detection system based on streaming technology. In *2014 Sixth International Conference on Intelligent Human-Machine Systems and Cybernetics (IHMSC)* (Vol. 2, pp. 275–279).