SMART: Simple Monitoring enterprise Activities by RFID Tags

Fabrizio Angiulli[2], Elio Masciari[1]

[1] ICAR-CNR – Institute of Italian National Research Council
masciari@icar.cnr.it
[2] DEIS-UNICAL
Via P. Bucci, 87036 Rende (CS) Italy
fangiulli@deis.unical.it

Abstract. Datastreams are potentially infinite sources of data that flow continuously while monitoring a physical phenomenon, like temperature levels or other kind of human activities, such as clickstreams, telephone call records, and so on. RFID technology has lead in recent years the generation of huge streams of data. Moreover, RFID based systems allow the effective management of items tagged by RFID tags, especially for supply chain management or objects tracking. In this paper we introduce SMART (Simple Monitoring enterprise Activities by RFID Tags) a system based on outlier template definition for detecting anomalies in RFID streams. We describe SMART features and its application on a real life scenario that shows the effectiveness of the proposed method for effective enterprise management.

1 Introduction

In this paper we will focus on Radio Frequency Identification (RFID) data streams monitoring as RFID based systems are emerging as key components in systems devoted to perform complex activities such as objects tracking and supply chain management. Sometimes RFID tags are referred to as electronic bar codes. Indeed, RFID tags emit a signal that contains basic identification information about a product. Such tags can be used to track a product from manufacturing through distribution and then on to retailers. These features of RFID tags open new perspectives both for hardware and data management. In fact, RFID is going to create a lot of new data management needs. In more details, RFID applications will generate a lot of so called "thin" data, i.e. data pertaining to time and location. In addition to providing insight into shipment and other supply chain process efficiencies, such data provide valuable information for determining product seasonality and other trends resulting in key information for the companies management. Moreover, companies are exploring more advanced uses for RFID. For instance, tire manufacturers plan to embed RFID chips in tires to determine the tire deterioration. Many pharmaceutical companies are embedding RFID chips in drug containers to better track and avert the theft of highly controlled drugs. Airlines are considering RFID-enabling key onboard

parts and supplies to optimize aircraft maintenance and airport gate preparation turnaround time.

Such a wide variety of systems for monitoring data streams could benefit of the definition of a suitable technique for detecting anomalies in the data flows being analyzed. As a motivating example you may think about a company that would like to monitor the mean time its goods stay on the aisles. Items are tagged by RFID tags so the reader continuously produces a readings that report the electronic product code of the item being scanned, its location and timestamp, this information can be used, as an example, for signaling that the item lays too much on the shelf since it is repeatedly scanned in the same position. It could be the case that the package is damaged and consequently customers tend to avoid the purchase. If an item exhibits such a feature it deserves further investigation. Such a problem is relevant to a so huge number of application scenario that it is impossible to define an absolute notion of anomalies (in the follow we refer to anomalies as outliers). In this paper we propose a framework for dealing with the outlier detection problem in massive datastreams generated in a network environment for objects tracking and management. The main idea is to provide users a simple but rather powerful framework for defining the notion of outlier for almost all the application scenarios at an higher level of abstraction, separating the specification of data being investigated from the specific outlier characterization.

2 Preliminaries

An RFID system consists of three components: the *tag*, the *reader* and the *application* which uses RFID data. Tags consist of an antenna and a silicon chip encapsulated in glass or plastic. RFID readers or receivers are composed of a radio frequency module, a control unit and an antenna to query electronic tags via radio frequency (RF) communication. They also include an interface that communicates with an application (e.g., the check-out counter in a store). Readers can be hand-held or mounted in specific locations in order to ensure they are able to read the tags as they pass through a *query zone* that is the area within which a reader can read the tag. The query zone are the locations that must be monitored for application purposes. In order to explain the typical features of an RFID application we consider the typical supply chain scenario.

The chain from the farm to the customer has many stages. At each stage goods are typically delivered to the next stage, but in some case a stage can be missing. The following three cases may occur: 1) the goods lifecycle begin at a given point (i.e. production stages, goods are tagged there and then move through the chain) and thus the reader in the zone register only departures of goods, we refer to this reader as *source reader*; 2) goods are scanned by the reader both when they arrive and they leave the aisle, in this case we refer to these reader as *intermediate reader*; 3) goods are scanned and the tag is killed, we refer to these readers as *destination reader*.

A RFID stream is (basically) composed of an ordered set of n sources (i.e., tag readers) located at different positions, denoted by $\{r_1, \ldots, r_n\}$ producing n independent streams of data, representing tag readings. Each RFID stream can be basically viewed as a sequence of triplets $\langle id_r, epc, \tau_s \rangle$, where: 1) $id_r \in \{1, .., n\}$ is the tag reader identifier (observe that it implicitly carries information about the spatial location of the reader) ; 2) *epc* is the product code read by the source identified by id_r and 3)τ_s is a *timestamp*, i.e., a value that indicates the time when the reading *epc* was produced by the source id_r.

An outlier is an observation that markedly differs from other observations as to lead to the suspect that it was generated by a different mechanism [4]. There exist several approaches to the identification of outliers, namely, statistical-based [2], distance-based [3], density-based [5] and MDEF-based [6]. The problem has been tackled from different viewpoint and in different scenarios such as static dataset, dynamic dataset and very large dataset[1]. In our application scenario we deal with massive datastreams that can be viewed as kind of a very large dynamic dataset. Based on the notion of RFID stream introduce so far, it is easy to see that each RFID reading generated by an RFID tag could be an outlier either because 1) the (product) features (obtained by the *epc* such as price, weight, height and so on) greatly differs from the others readings or 2) the latency time that the tagged item spent in a given location deviates significantly from an expected value.

In our system we will assume either distance based outlier function or statistical based outlier function to catch both source of anomaly and since we are interested in the problem formalization, we disregard here the actual outlier function implementation. More formally, given a set of objects S, a positive integer k, and a positive real number R. An object $o \in S$ is a $DB(k, R)$- outlier, or a distance-based outlier with respect to parameters k and R, if less than k objects in S lie within distance R from o. This kind of function will be exploited when searching for outliers based on their product features. To deal with deviation on time features we resort to statistical based outlier function. We point out that a formal analysis of the possible outlier detection methods is out of the scope of this paper, we mentioned here the main approaches used in literature since in our system implementation we allow any stream oriented implementation of outlier function to be used. The latter observation guarantees a high flexibility in our system for dealing with every possible application scenarios.

3 Statement of the Problem

In our model, *epc* is the identifier associated with a single unit being scanned (this may be a pallet or a single item, depending on the level of granularity chosen for tagging the goods being monitored).

This basic schema is simple enough to be used as a basic schema for a data stream environment, anyway since more information are needed about the outlier being detected we can access additional information by using some auxiliary tables maintained at a *Master* site as shown in figure 2. More in detail, the

Master maintains an intermediate local warehouse of RFID data that stores information about items, items' movements, product categories and locations and is exploited to provide details about RFID data upon user requests. The information about items' movements are stored in the relation $ItemMovement$ and the information about product categories and locations are stored in the relations *Product* and *Locations*, respectively. These relations represents, respectively, the *Product* and the *Location* hierarchy. Relation *EPCProducts* maintains the association between *epcs* and product category, that is, every *epc* is associated to a tuple at the most specific level of the *Product* hierarchy. Finally, RFID readers constitute the most specific level of the *Location* hierarchy.

$ItemMovements$ contains tuples of the form $\langle epc, DL \rangle$, where *epc* has the usual meaning, and DL is string built as follows: each time an *epc* is read for the first time at a node N_i a trigger fires and DL is updated appending the node identifier.

In the following we define a framework for integrating DSMS technologies and outlier detection framework in order to effectively manage outliers in RFID datastreams. In particular we will exploit the following features: *a)* The definition of a template for specifying outlier queries on datastreams that could be implemented on top of a DSMS by mapping the template in a suitable set of continuous queries expressed in a continuous query language language ESL-like[7]; *b)* The template need to be powerful enough to model all the interesting surveillance scenarios. In this respect, it should allow the definition of four components, namely: 1) the kind of objects (O) to be monitored (e.g. RFID data concerning dairy products),2) the reference population P (due to the infinite nature of datastream) depending on the application context (e.g. a subset of the items belonging to dairy products category), 3) the attributes (A) of the population used for signing out anomalies (e.g. time spent at a given location), 4) the outlier definition by means of a suitable function $\mathcal{F}(P, A, O) \rightarrow \{0, 1\}$ (e.g. deviation from the average time spent at a given location by an item); *c)* A mapping function that for a given template and DSMS schema, resolve the template in a set of outlier continuous queries to be issued on the datastream being monitored.

The basic intuition behind the template definition is that we want to run an aggregate function that is raises by the *Master* (that is a central node collecting the queries and the aggregate statistics along with the sample populations) and then instantiated on a subset of nodes in the network. An incoming stream is processed at each *node* where the template is activated by the $Master$ that issue the request for monitoring the stream. Once a possible outlier is detected, it is signaled to the $Master$. The master maintains management information about the network and some additional information about the items using two auxiliary tables $OutlierMovement$ and $NewTrend$. In the $OutlierMovement$ table it stores information about the outlying objects, in particular it stores their identifiers and the paths traveled so far as explained above for $ItemMovements$. The $NewTrend$ table stores information about objects that are not outliers but instead they represent a new phenomenon in the data. It contains tuples of the

form $\langle epc, N, \tau_a, \tau_l, \rangle$, where N is a node, τ_a and τ_l are, respectively, the arrival time and the time interval spent at node N by the *epc*. The latter table is really important since it is intended to deal with the concept drift that could affect the data. Indeed, when items are marked as unusual but they are not an anomalies as in the case of varied selling rates they are recorded for later use in outlier definition. In particular, once the new trend has been consolidated, new statistics for the node where the objects appeared will be computed at $Master$ level and then forwarded to the pertaining node in order to update the parameters of its population.

As mentioned above candidate outliers are signaled at node level but they are managed by the master. More in detail, as a possible outlier is signaled by a given node the master stores it in the $OutlierMovement$ table along with its path if it is recognized as an anomaly or in the $NewTrend$ table if a signaled item could represent the symptom of a new trend in data. To summarize, given a signaled object o two cases may occur: 1) o is an outlier and then it is stored in the $Outlier$ table; 2) o represent a new trend in data distribution and then it should not be considered an outlier and we store it in the $NewTrend$ table. To better understand such a problem we define three possible scenarios on a toy example.

Example 1. Consider a container (whose *epc* is p_1) containing dangerous material that has to be delivered through check points c_1, c_2, c_3 in the given order and consider the following sequence of readings: $Seq_A = \{(p_1, c_1, 1), (p_1, c_1, 2), (p_1, c_2, 3), (p_1, c_2, 4), (p_1, c_2, 5), (p_1, c_2, 6), (p_1, c_2, 7), (p_1, c_2, 8), (p_1, c_2, 9), (p_1, c_2, 10), (p_1, c_2, 11), (p_1, c_2, 12)\}$. Sequence A correspond to the case in which the pallet tag is read repeatedly at the check point c_2. This sequence may occur because: i) the pallet (or the content) is damaged so it can no more be shipped until some recovery operation has been performed, ii) the shipment has been delayed. Depending on which one is the correct interpretation different recovery action need to be performed. To take into account this problem in our prototype implementation we maintain appropriate statistics on latency time at each node for signaling the possible outlier. Once the object has been forwarded to the master a second check is performed in order to store it either in $OutlierMovement$ or in $NewTrend$ table. In particular, it could happen that due to new shipping policy additional checks have to be performed on dangerous material, obviously this will cause a delay in shipping operations, thus the tuple has to be stored in the $NewTrend$ table.

Consider now a different sequence of readings: $Seq_B = \{(p_1, c_1, 1), (p_1, c_1, 2), (p_1, c_1, 3), (p_1, c_1, 4), (p_1, c_3, 5), (p_1, c_3, 6), (p_1, c_3, 7), (p_1, c_3, 8), (p_1, c_3, 9), (p_1, c_3, 10), (p_1, c_3, 11), (p_1, c_3, 12)\}$. Sequence B correspond to a more interesting scenario, in particular it is the case that the pallet tag is read at check point c_1, is not read at check point c_2 but is read at checkpoint c_3. Again two main explanation could be considered: i) the original routing has been changed for shipment improvement, ii) someone changed the route for fraudulent reason (e.g. in order to steal the content or to modify it). In this case suppose that the shipping plan

has not been changed, this means that we are dealing with an outlier then we store it in the $OutlierMovement$ table along with its path.

Finally, consider the following sequence of readings regarding products p_1, p_2, p_3 that are frozen foods, and product p_4 that is perishables, all readings generated at a freezer warehouse c: $Seq_C = \{(p_1, c, 1), (p_2, c, 2), (p_3, c, 3), (p_4, c, 4), (p_1, c, 5), (p_2, c, 6), (p_3, c, 7), (p_4, c, 8), (p_1, c, 9), (p_2, c, 10), (p_3, c, 11), (p_4, c, 12)\}$. Obviously, p_4 is an outlier for that node of the supply chain and this can be easily recognized using a distance based outlier function since its expiry date greatly deviates from the expiry dates of other goods.

The Template in a short In this section we will describe the functionalities and syntax of the $Template$ introduced so far. A $Template$ is an aggregate function that takes as input a stream. Since the physical stream could contain several attributes as explained in previous sections we allow *selection* and *projection* operation on the physical stream. As will be clear in next section we will use a syntax similar to ESL with some specific additional features pertaining to our application scenario. This filtering step is intended for feeding the reference population P. In particular, as an object is selected at a given node it is included in the reference population for that node using an $Initialize$ operation, it persists in the reference population as a $Remove$ operation is invoked (it can be seen as an $Initialize$ operation on the tuples exiting the node being monitored).

We recall that a RFID tagged object is scanned multiple times at a given node N so when the reader no more detects the RFID tag no reading is generated. First time an object is read a $Validate$ trigger fires and send the information to the $Master$ that eventually updates the $ItemMovement$ table. In response to a $Validate$ trigger the $Master$ performs a check on the item path, in particular it checks if shipping constraints are so far met. In particular, it checks the incoming reading for testing if the actual path so far traveled by current item is correct. This check can be performed by the following operations: 1) selection of the path for that kind of item stored in $ItemMovement$, 2) add the current node to the path, 3) check the actual path stored in an auxiliary table $DeliveryPlans$ storing all the delivery plans (we refer to this check as $DELIVERY\ CHECK$). This step is crucial for signaling path anomalies since as explained in our toy examples that source of anomaly arise at this stage. If the item is not validated the $Master$ stores the item information in order to solve the conflict, in particular it could be the case that delivery plans are changing (we refer to this check as $NEW\ PATH\ CHECK$) so information is stored in $NewTrend$ table for future analysis , otherwise it is stored in the $OutlierMovement$ table. To better understand this behavior consider the Seq_B in example 1. When the item is first time detected at node c_3 the $Validate$ trigger fires, the path so far traveled for that object is retrieved obtaining $path = c_1$, the current node is added thus updating $path = c_1.c_3$ but when checked against the actual path stored in $DeliveryPlans$ an anomaly is signaled since it was supposed to be $c_1.c_2.c_3$. In this case the item is stored in the $OutlierMovement$ table and the $Master$ signal for a recovery action. It works analogously for Seq_A as explained in example 1.

When an *epc* has been validated it is added to the reference population for that node (P_N) then it stays at the node and is continuously scanned. It may happen that during its stay at a given node an *epc* could not be read due to temporary field problem, we should distinguish this malfunction from the "normal" behavior that arise when an item is moved for shipping or (in case of destination nodes) because it has been sold. To deal with this feature we provide a trigger $Forget$ that fires when an object is not read for a (context depending) number of reading cycles (we refer in the following as *TIMESTAMP CHECK*. We point out that this operation is not lossy since we recall that at each node we maintain (updated) statistics on items. When $Forget$ runs, it removes the "old" item from the actual population and update the node statistics. Node statistics (we refer hereafter to them as $model_M$ where N is the node they refer to) we take into account for outlier detection are: number of items grouped by product category (*count*), average time spent at the node by items belonging to a given category (m), variance for items (v) belonging to a given category, maximum time spent at the current node by items belonging to a given category (max_t), minimum time spent at the current node by items belonging to a given category (min_t). By means of the reference population P_N and the node statistics $model_N$ the chosen outlier function checks for anomalies. In particular, we can search for two kind of anomalies: 1) *item based* anomalies, i.e. anomaly regarding the item features, in this case we will run a distance-based outlier detection function; 2) *time based* anomalies, i.e. anomaly regarding arrival time or latency time, in this case we will run a statistical based outlier detection function.

3.1 The RFID-$\mathcal{T}$ syntax

In this section we formalize the syntax for template definition. For basic stream operation we will refer to ESL-like syntax[7]. We point out that even if in this paper we focus on RFID data and outlier detection task, the framework is rather general and could be exploited in several application domains and for other task such as aggregate queries evaluation.

The first step is to create the stream related to nodes being monitored. Once the streams are created at each node the $Template$ definition has to be provided.

Aggregate function can be any SQL available function applied on the reference population as shown in Fig. 5, where $Return$ and $Next$ have the same interpretation as in SQL and $< Type >$ can be any SQL aggregate function. An empty $TERMINATE$ clause refer to a non-blocking version of the aggregate.

As the template has been defined it must be instantiated on the nodes being monitored. In particular triggers $Validate$ and $Forget$ are activated at each node. As mentioned above they will continuously update the reference population and node and $Master$ statistics. The syntax of these triggers is shown in figure 6.

We point out again that $Validate$ trigger has the important side-effect of signaling *path* outliers. We point out that the above presented definition is completely flexible so if the user may need a different outlier definition she simply needs to add its definition as a plug-in in our system.

CREATE STREAM	$< name >$
ORDER BY	$< attribute >$
SOURCE	$< systemnode >$
DEFINE OUTLIER TEMPLATE	$< name >$
ON STREAM	$< streamname >$
REFERENCE POPULATION	$(< definepopulation >)$
MONITORING	$(< target >)$
USING	$< outlierfunction >$
$< definepopulation >$	INSERT INTO $< PopulationName >$ SELECT $< attributelist >$ FROM $< streamname >$ WHERE $< conditions >$
$< target >$	$< attributelist > \mid < aggegatefunction >$
$< outlierfunction >$	$< distancebased > \mid < statisticalbased >$

Fig. 1. Template Definition syntax

AGGREGATE	<Function Name> <Type>(Next Real) : Real
{ TABLE	<Table Name> (<attribute list>);
INITIALIZE:	{ INSERT INTO <Table Name> VALUES (Next, 1); }
ITERATE:	{ UPDATE <Population Name> SET <update condition>;
INSERT INTO RETURN	SELECT <output attribute> FROM <Table Name> }
TERMINATE :	{} }
CREATE TRIGGER	Validate
BEFORE	INSERT ON <Population Name>
REFERENCING NEW AS	NEW READING
IF PATH CHECK	INSERT INTO <Population Name> VALUES (NEW READING)
ELSE IF DELIVERY CHECK	INSERT INTO NewTrend VALUES (NEW READING)
ELSE	INSERT INTO OutlierMovement VALUES (NEW READING)
CREATE TRIGGER	Forget
AFTER	INSERT ON <Population Name>
REFERENCING OLD AS	OLD READING
IF TIMESTAMP CHECK	{DELETE FROM <Population Name> OLD READING
	UPDATE STATISTICS ON <Population Name> }

Fig. 2. Aggregate Function, Validate an Forget trigger syntax

References

1. F. Angiulli and F. Fassetti. DOLPHIN: An Efficient Algorithm for Mining Distance-Based Outliers in Very Large Datasets. *TKDD*, 8(1), 2009.
2. V. Barnett and T. Lewis. *Outliers in Statistical Data*. Wiley and Sons, 1994.
3. R. NG E. Knorr and V. Tucakov. Distance-based outlier: Algorithms and applications. *VLDB J.*, 8(3-4):237–253, 2000.
4. D. Hawkins. *Identification of Outliers*. Monographs on Applied Probability and Statistics. Chapman and Hall, 1980.
5. R. Ng J. Sander M.M. Breunig, H. Kriegel. Lof: Identifying density-based local outliers. In *In Proceedings of the International Conference on Managment of Data (SIGMOD00)*.
6. P. Gibbons S. Papadimitriou, S. Kitagawa and C. Faloutsos. Loci: Fast outlier detection using the local correlation integral. In *In Proceedings of the International Conference on Data Enginnering (ICDE)*, pages 315–326, 2003.
7. H. Thakkar H. Wang Y. Bai, R.C. Luo and C. Zaniolo. An introduction to the Expressive Stream Language (ESL). *Tech. Report*.