SysRFID: generation of synthetic data in Supply Chains

Roberto De Virgilio

Abstract Recently, due to commercial success of RFID technology, there has been much attention to adopt such technologies in supply chain data management. This motivates the proliferation of massive data analysis techniques of a particular complexity supported of proper data management and aggregation schemes in such systems. Nevertheless testing these frameworks is a complex task since they are not able to easily obtain sensitive information due to security, privacy or cost issues. This paper describes a platform for the generation of realistic synthetic RFID data that can facilitate the development and testing of data mining tools for supply chain management. In particular we present SysRFID, a framework that allows to simulate the output you would generate in a supply-chain where the RFID technology is used. In the paper it is shown how to set the environment and the execution steps to obtain an output strictly related to required needs. Finally experimental results demonstrates the feasibility of the system.

1 Introduction

A supply chain is a complex system composed of organizations and people with their activities involved in transferring a product or service from a supplier to a final customer. In such scenario, Radio-Frequency IDentification (RFID) is a promising infrastructure-less technology interconnecting via radio two main components: (1) transponders carrying data (tags), located on the objects to be identified; (2) interrogators (readers) able to receive the transmitted data. Traditional RFID applications have been focused on replacing bar code technology in supply chain management and asset tracking [6]. Benefits introduced by RFID technology w.r.t. barcodes include: (i) line-of-sight is not required between reader and tag, unlike optical scan; (ii) larger read range (up to few meters); (iii) nearly simultaneous detection of multiple RFID tags; (iv) higher tag storage capacity. Because of these features, RFID provides higher levels of automation in the supply chain and helps prevent human errors (e.g. a reader can inventory an entire shipment in one pass while it is loaded into a warehouse, without having to scan each

Dipartimento di Informatica e Automazione
Universitá Roma Tre, Rome, Italy
e-mail: dvr@dia.uniroma3.it

product). In latest years, industry is progressively rallying around few worldwide standards for RFID technologies. In this effort a leading role is played by the EPCglobal consortium (`http://www.epcglobalinc.org.`). In Supply Chain Management (SCM) tools based on RFID technology, the practical issue of obtaining the data sets to be integrated with these tools remains an obstacle. The IT industry needs synthetic data generators for different applications including regression testing (i.e. repeatedly generate the same large data set for testing enterprise applications), secure application development (i.e. testing on realistic but not real data) and data mining testing (i.e. generate data sets with known characteristics to gauge whether data mining tools can discover those characteristics). Existing synthetic data generation tools in the commercial world can generate modest amounts of easily described data: they do not scale for generating industrial-sized (i.e., terabyte) data sets.

To address this issues, in this paper we illustrate the development of SYS-RFID a platform for the generation of large amount of realistic synthetic RFID data efficiently. Realistic synthetic data can serve as background data sets into which SCM tools require substantially testing and revisions of analysis in order to maximize efficient performance and accuracy that could be very costly and hypothetical future scenarios can be overlaid.

2 State of the art

There are several commercial systems, like TURBO (`http://www.turbodata.ca`), GS (`http://www.GSApps.com`), DTM (`http://www.sqledit.com`), ROWGEN (`http://www.iri.com`) which allow users to generate large sets of data for performance and scalability testing of DB and DW applications. They generate "meaningful" data, in the sense that they generate random data of appropriate type, using functions, built-in dictionaries, custom lists, dependencies between columns, complex rules. Only GS can express time dependency between sequential business events, which is a crucial feature to generate "realistic" RFID data derived from a supply chain. Anyway, even with a time dependency management feature, generating RFID data is still a time-consuming job, for the complex dependencies which there are among object stocks, locations, and transfer speed for different object categories. In [1] attention is focused on data generation with complex statistical distribution, but events and timestamps are not managed in the sophisticated way we need. In [2] authors use a graph model to simplify the expression of complex inter and intra table relationships. Some interesting ideas are in [3, 5]. The authors provide *Information Discovery and Analysis Systems*(IDAS) that uses data mining methodologies to identify significant events and relationships, but a critical technical issue is testing their ability to provide accurate inference. An IDAS Data and Scenario Generator (IDSG) is proposed. Knowledge about object attributes and their relationships is represented though a semantic graph, and the generation function for an attribute (a node) takes the values of the incoming links as input. The generation function may depend

on one or more statistical distributions over a set of discrete or continuous values. This way to express dependencies among data can be useful to generalize the RFID data generation, but only if we describe some variables, e. g. reading timestamps for a same object in different locations, with different attributes (nodes). Finally, in [4] several techniques for very large databases are presented: parallelism to speedup and scale-up the generation, concurrent generation of indices, and special statistical distributions of data. The goal is to facilitate the generation of a large amount of data, not the creation of data-sets for a specific area as the RFID supply chain.

3 Preliminaries

RFID data modeling. An RFID application usually generates a stream of RFID tuples of the form of a triple $\langle E, l, t\rangle$, where: (i) E is an EPC (Electronic Product Code), a unique identifier stored in an RFID tag and associated with the object carrying the tag, (ii) l represents the location where an RFID reader has scanned an object having E as EPC, and (iii) t is the time when the reading took place. This type of stream is usually called *raw data*. Other properties of the reading event can be retrieved like the temperature, the pressure and the degree of humidity. Typically such properties are encoded (in case) into the EPC. A single tag may have multiple readings at the same location, thus producing a great amount of raw data. Therefore, a simple cleaning technique consists of converting raw data in *stay records* of the form: $\langle E, l, t_{in}, t_{out}\rangle$ where t_{in} is the time when the object enters the location l, and t_{out} is the time when the object leaves the location. This basic compression is the largely used and reduces the amount of data to be stored, although not relevantly.

Supply Chain Modeling. The basic idea is to represent a supply chain s by a directed acyclic graph G_s (that we call *sc-graph*), in which the nodes represent the locations of s and there is an edge from a node l_1 to a node l_2 if there is some movement of objects from l_1 to l_2 in s. The *source nodes* of G_s, i.e. the nodes having no incoming edge, usually represent the place in which the objects of the supply chain are produced, whereas the *target nodes*, i.e. the nodes having no outgoing edge, are usually the final stores where the objects sold.

4 RFID data configuration

Our system generates the output file on the basis of some parameters indicated in a configuration file (i.e. serialized in XML). In such file it is possible to configure the structure of the supply chain, to set different *categories* (i.e. properties) which will refer to the stocks of items, and others general setting supporting the simulation process.

Items Categorization. For item we mean an object with an RFID tag. An item can be a single object (e.g. a pencil) or a set of objects (e.g a pack of 6 pencils). In the latter case the item presents a unique EPC but we associated the number of contained objects. For instance it is reasonable to track a stock of bottles as a single item with a unique EPC instead of considering individually different items with corresponding EPCs. In the simulation, SysRFID focuses on tracking the flow of items into the sc-graph. The items are characterized by categories. Each category has its own characteristics that distinguish it from each others. Therefore in the configuration file an element <category> will contain several attributes. For instance let's consider the following example

```
<categories>
 <category
    name = "PC" item_count = "1500" branch_rate = "1"
    level_branch_factor = "1" handling_time = "4"
    unit_time = "hh" unique_items = "true"
    price = "850.0" model = "Xeon 3200" brand = "IBM" />
...
</categories>
```

The attribute *Name* is an identification name of the category. *Item Count* is the amount of items in the category produced by the system. *Branch Rate* is the amount of items that move together from a location to another (in the example the items move singularly). *Level Branch Factor* is a correction factor for branch rate depending on the depth of the location in the graph. It allows to create smaller stocks as it descending into deeper levels of the graph. *Handling time* is the average time of permanence of the item in a location. *Unit time* is the time unit for the handling time: ss (seconds), hh (hours) and dd (days). *Unique items* is a boolean value that is true if the item is a single object, false if the item is a set of objects (i.e. a stock) that share the same EPC. *Price, Model* and *Brand* are properties of each item (i.e. easily extensible).

Supply Chain Configuration. As described above, we model the supply chain as an sc-graph. At start-up we have to generate a new sc-graph. The generation of the graph is performed in two distinct ways: *fixed chain* and *random chain*. In the former, we define manually the list of locations (i.e. nodes) and the list of connections between them (i.e. edges). For instance an example follows

```
<chain type="fixed">
 <location name = "L0" distance = "13">
    <location name = "L11" distance = "20">
       ...
     </location>
   </location>
</chain>
```

In such file the nested composition of two locations is the parent relationship. Moreover for each location we define the *distance* by the parent (i.e. this parameter is needed to calculate the time of moving from one location

to another). In the random chain we have to define the number of levels, the maximum number of nodes for each levels and the maximum output cardinality of each node. The sum of cardinalities from each location at the same level has to be less or equal to the number of nodes generated in the current level. Then the system will generate automatically (random) the sc-graph from that settings (i.e. locations, connections and distances). Let's consider the following example

```
<chain type="random">
 <location level = "1" nodes = "1" out = "19"/>
 <location level = "2" nodes = "19" out = "17"/>
 ...
</chain>
```

To our sc-graph (both in fixed and random chain) the system automatically adds a fictitious location that we call *super-root*, that is the parent of all sources. This location represents the big *warehouse* where all items are generated and from which such items are disseminated along the chain. In the simulation each location L has two readers L_{in} and L_{out} that record when an item goes into a location and leaves the same location respectively. The sensors have a read speed not infinite. Each reader is able to scan a number of items at the same time t_0 with respect to a configurable capability value (i.e. 100 is the default value). For instance, given the capability 100 and an incoming stock of 115 items, the reader records 100 items at time t_0 and the others 15 at time t_1.

Simulation Settings. We can define some basic settings for our simulation. Therefore in the configuration file an element <simulation> will contain several attributes. The attribute *Records* defines the type of record to write into the output: Raw data (rd) or Stay Record (sr). *Workday hours* describes how many hours you consider into a working day. *Gap reads* is the capability of a reader to trace items at the same time (the default value is 100 items). *Offset* is the starting timestamp of the simulation. *Window* is the time window where generated readers are considered and included into the output. It is expressed in hours and starts from the offset. *Output* is the format of final output. The system can provide structured files (i.e. XML and CSV): in this case we can have a single big file or multiple files, each one corresponding to records into a single location. Moreover the records can be stored also into a DBMS (i.e. actually PostgreSQL or MySQL). Finally *src* describes the path to write the file or the connection string for the DBMS.

5 RFID data generation

The start-up of the simulation prepares the chain (i.e. if it is random, it will be generated) and then generates all the items in the super-root node (i.e. the products will start to move from the warehouse location) with respect to the configuration file. Each location has a level, that is the depth from the warehouse node, and each edge is labelled by the distance between the

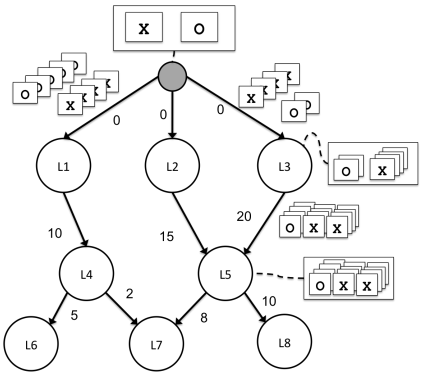

Fig. 1 An example of simulation

referring locations. An object `StockBucket` contains all items grouped in stocks with respect to the corresponding category. At the beginning of the simulation, intermediate locations are empty. The stocks of items will be splitted (according to the rate defined in the configuration file) when they leave a location L_i to enter into another location L_j. In this case we assume that L_i is parent of L_j and the level of L_j in the graph is higher than the level of L_i, that is we do not allow cycles or movements at the same level in the graph. The simulation will finish when all items are sold out, that is all products reach the sinks of the graph (i.e. the final shops of the supply chain). We remind that a window of *useful* time (defined in the configuration file) allows to consider only readings into a temporal range (in case), therefore selecting only the transactions of interest. According to the configuration file, if the type of records to generate is raw data, for each item that remains in the same location we generate multiple readings at fixed time intervals. Otherwise, if we have stay records, we generate only the readings when the item enter and leaves a location.

Execution. The `StockBucket` is organized as a queue. Each transaction extracts the top element (i.e. a stock) from `StockBucket` and splits it in several sub-stocks st_i. Then each st_i enters into a location L_i directly connected to the warehouse, and we register the time t_{in}. The items of st_i will remain into L_i for a time t_h that is the handling time of the corresponding category. When the items of st_i have to leave L_i, each st_i is splitted into several sub-stocks st_j registering a time $t_{out} = t_{in} + t_h$. Then each st_j will enter into a location L_j directly connected to L_i. Of course the time t'_{in} in L_j will be $t_{in} + t_d$ where t_d is the time to cover the distance between L_i and L_j. The process goes on similarly. Each location presents a `StockBucket`. Therefore, when different stocks (i.e. stocks corresponding to different categories) enter into a location, that stocks are inserted into the local queue and processed with respect to the order of arrival. Each stock is splitted according to the following: `(item_count * branch_rate) / (level_branch_factor * level)`.

For instance let us consider the example of Fig. 1. We have two categories (*i.e.*x and ∘) with the same configuration as follows:

```
Item_count = 1000
Branch_rate = 0.1
Level_branch_factor = 1
```

This means that initially we have a stock of 1000 items for category. Since the warehouse is in the level 1 in the chain, we split the stock in blocks of 100 items (i.e $1000 * 0.1$ / $1 * 1$). In the next shipment (i.e. level 2) each stock of 100 items will be splitted in blocks of 50 items (i.e $1000 * 0.1$ / $1 * 2$). The next blocks will contain 25 items (i.e $1000 * 0.1$ / $1 * 3$), and so on.

6 Implementation

We have implemented SysRFID, a Java tool for the generation of realistic synthetic RFID data. The tool is according to a client-server architecture. At client-side, we have a Web interface (as shown in Fig. 2) based on GWT[1] that provides support for filling the configuration file and launching of the generation. At serve-side, we have the core of SysRFID and the generation of the required dataset. In this way we have a Web service where the communication is asynchronous: the user launch the generation process and the server produces the required data, store locally such data and sends a message to the user by email containing a link to download the data on the client side.

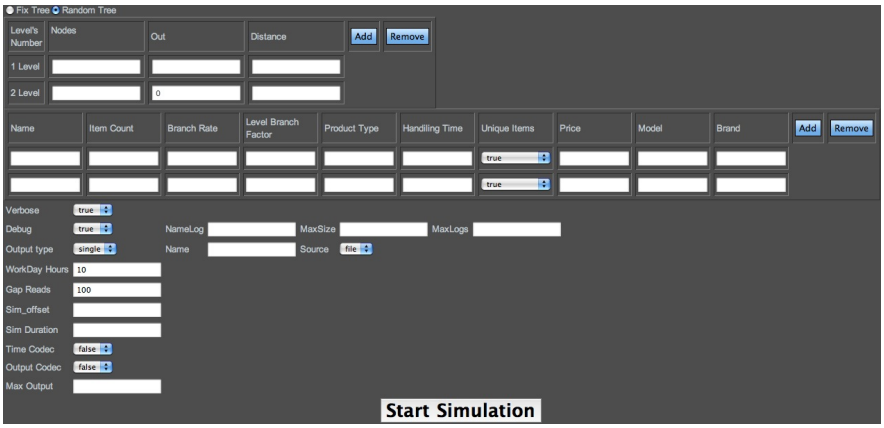

Fig. 2 Front-end of SysRFID: Configuration and Launch of the simulation

We are developing experimental interface to follow interactively the process at running on the fly. We have executed several experiments to test the performance of SysRFID. We used a dual core 2.66GHz Intel with 2 GB of

[1] GWT, `http://code.google.com/webtoolkit/`

main memory running on Linux. We configured a supply chain of 100 locations with 8 levels, four categories of items, totally 100000 items, branch_rate = 0.1 and level_branch_rate = 1 for each category, and a unique CSV file as output format. We executed the simulation to generate different datasets of both raw data (RD) and stay records (SR). Fig. 3 shows the resulting response times. We evaluated the time (sec) to generate 10^5, 10^6, 10^7 and 10^8 readings. The results demonstrate the efficiency (and scalability) of SysRFID to generate large amount of data (e.g. 10^8). Of course the generation of raw data requires a more expensive processing, since each transaction has to manage multiple readings in each location.

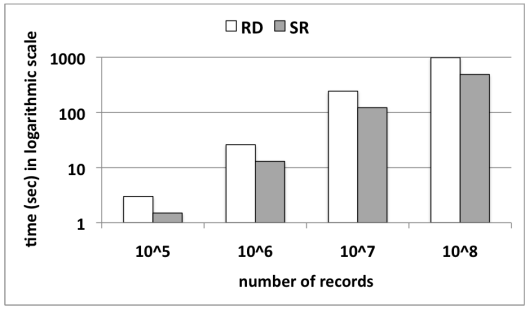

Fig. 3 Efficiency of SysRFID

7 Conclusion

We have proposed a platform for the generation of synthetic RFID data that can facilitate the development and testing of data mining tools for supply chain management. We implemented such framework in SysRFID, a Java system that allows to simulate the output you would generate in a supply-chain where the RFID technology is used. We have shown how to set the environment and the execution steps to obtain an output strictly related to required needs.

References

1. N. Bruno and S. Chaudhuri. Flexible Database Generators. In *VLDB, pp. 1097-1107*, 2005.
2. K. Houkjaer, K. Torp and R. Wind. Simple and Realistic Data Generation. In *VLDB, pp. 1243-1246*, 2006.
3. P. Lin et al. Development of a Synthetic Data Set Generator for Building and Testing Information Discovery Systems. In *ITNG, pp. 707-712*, 2006.
4. J. Gray et al. Quickly generating billion-record synthetic databases. In *SIGMOD, pp. 243-252*, 1994.
5. D. R. Jeske et al. Generation of Synthetic Data Sets for Evaluating the Accuracy of Knowledge Discovery Systems. In *SIGKDD, pp. 756-762*, 2005.
6. R. Weinstein. RFID: A Technical Overview and Its Application to the Enterprise. In *IT Professional, 07(3):2733*, 2005.