A Prediction-based Smart Meter Data Generator

Nadeem Iftikhar*, Xiufeng Liu†, Finn Ebertsen Nordbjerg*, Sergiu Danalachi*
*University College of Northern Denmark
Sofiendalsvej 60, 9200 Aalborg SV, Denmark
{naif,fen,1028752}@ucn.dk
†Technical University of Denmark
Building 426, 2800 Kgs. Lyngby, Denmark
xiuli@dtu.dk

***Abstract*—With the prevalence of cloud computing and Internet of Things (IoT), smart meters have become one of the main components of smart city strategies. Smart meters generate large amounts of fine-grained data that is used to provide useful information to consumers and utility companies for decision-making. Now-a-days, smart meter analytics systems consist of analytical algorithms that process massive amounts of data. These analytics algorithms require ample amounts of realistic data for testing and verification purposes. However, it is usually difficult to obtain adequate amounts of realistic data, mainly due to privacy issues. This paper proposes a smart meter data generator that can generate realistic energy consumption data by making use of a small real-world data set as seed. The generator generates data using a prediction-based method that depends on historical energy consumption patterns along with Gaussian white noise. In this paper, we comprehensively evaluate the efficiency and effectiveness of the proposed method based on a real-world energy data set.**

***Keywords*-smart meter; data generator; time-series**

I. INTRODUCTION

A smart meter is an electronic device that records consumption of energy in intervals of an hour or less. It communicates with energy management system (EMS) at regular time-intervals to transmit meter readings [1]. Smart meter data is processed and analyzed by EMS to provide decision support to consumers and utility companies. For example, to benefit customers to save energy and to help utility companies to accurately forecast energy demand. Further, the development of smart meter analytics systems, algorithms and applications is extremely dependent on the availability of realistic data for training and testing purposes. The realistic data is also necessary for discovering new insights, debugging and identifying performance bottlenecks. Thus, ample amounts of realistic data can ensure that the analytics systems can provide the required accuracy and performance. In reality, it is usually difficult to obtain sufficient amounts of real-world smart meter data to test analytics systems and applications. This is primarily due to legal reasons, business secrets and/or data privacy issues. On the other hand, developers need test data, hence they ought to create data that is best suited to their needs. Typically, the developers generate data either by replicating the available data or by writing data generation script for each use case. Still, the generated data is not able to reflect the characteristics (for example, trend or pattern) of the actual data. To solve these issues, a synthetic data generator is necessary to test data analytics systems and algorithms. In this paper, we present a prediction-based data generator that can generate realistic smart meter data. The data generator takes a real-world energy consumption time series as seed. Based on the historical patterns of energy consumption, it generates synthetic time series data sets. In doing so, the generator first creates an adjusted time series by reducing the periodic variations from the actual time series (smooths morning/evening peak periods). Afterwards, it uses a prediction-based model (*P*eriodic Autoregressive (PAR) model [2]) to predict meter readings. In the end, the periodic variations are added back to the forecast meter readings to reflect the pattern and variance of the real-world consumption time series. Although in this paper, we use the Irish data set [3] as seed, however, the data generator can take other similar real-world time series as seed.

Our main contributions are the following: 1) we propose a smart meter data generator that has the ability to generate synthetic smart meter time series data based on randomly selected consumption patterns; 2) we present a statistical-based model to generate synthetic time series data; 3) we propose a smart meter data aggregation solution to gradually aggregate time series data at different granularities; and 4) we suggest a possibility to generate random accelerated peaks in order to test and verify anomaly detection algorithms.

The paper is structured as follows. Section II presents the background and requirements. Section III describes the proposed method. Section IV evaluates the method, followed by presenting the related work in Section V. We conclude the paper and point out the future research directions in Section VI.

II. BACKGROUND AND REQUIREMENTS

In this section, we present the background knowledge of smart meter data and the requirements for implementing the data generator. We use the smart meter data set from Irish Social Science Data Archive (ISSDA) [3], as an example to explain the time series consumption behavior. Within the Irish data set, we choose the electricity consumption data that is collected from over 5,000 homes and businesses. These homes and businesses participated in the smart metering electricity customer behaviour trials between 2009 and 2010. The original Irish time series data is at a 30-minute resolution, however, we aggregate the time series to an hourly resolution. The modified Irish time series has the following format: <*MeterID, Timestamp, Value*>. Meter ID represents a particular meter installed at a specific location. Timestamp is the point in time at which a reading is recorded. Value is the electricity consumption in kWh collected every 60 minutes.

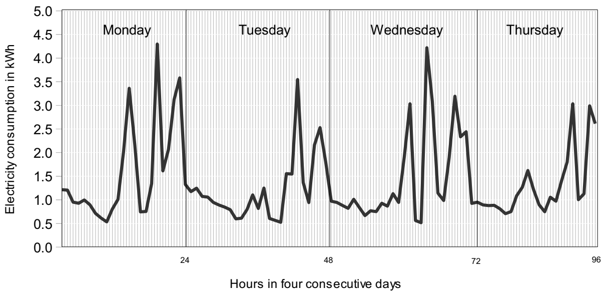


Fig. 1. Weekdays consumption pattern of a typical private household

In order to understand the background and requirements, we have selected a time series of energy consumption of a private household (Figure 1). We have observed that this household has roughly a fixed consumption pattern during weekdays. For example, the household does not use much energy in the morning (unlike many other households which have a morning peak). In the afternoon, there is a slight increase in the consumption of energy between 13:00 and 15:00 o'clock, which is probably the time when children come back from school. There is also a considerable increase in the consumption of energy between 18:00 and 23:00 o'clock that is the time when all the household is at home. During this time, the household uses electricity for lighting, cooking, washing and so on. According to the information provided by the Irish data, the household has the following profile: the head of household belongs to the age group between 36-45; the chief income earner in the household is working as an employee; the household consists of both adults and children (under 15 years of age); the household is away for at least 5-6 hours during the day; the household uses oil to heat their home; and uses electricity for cooking. Further more, the household has following appliances: a washing machine; a tumble dryer; a dishwasher; an electric shower; an electric heater (plug-in) and so on. In a typical day, the household uses washing machine, tumble dry and dishwasher for less than an hour each, and electric cooker for 30-60 minutes. The household has many entertainment appliances, including a television (TV) less than 21 inch, two TV's greater than 21 inch, three desktop computers and one game console (Xbox). The daily use of 2 TV's greater than 21 inch is more than 5 hours, desktop computer between 1-3 hours, Xbox between 3-5 hours and so on. So, we can observe that the energy consumption of the household follows a certain pattern, which is closely related to the living habits of the household and their use of appliances. Based on this observation, it seems feasible, to generate data according to the consumption pattern of customers.

III. The Proposed Method

A. Overview

In this section, we present the main idea behind the proposed data generator. We decide to use the quantitative model approach, as this approach is expressed in mathematical notation and based on data quantity. The quantitative model is further divided into casual model and time-series model. We choose the latter for modeling time series data for energy consumption. The time-series model produces forecast by making use of historical behavior of values. A time series may also have trend, cyclic pattern and/or seasonal pattern. The seasonal/periodic pattern normally exists when a series influences with periodical factors, such as weekday/weekend and so on. The periodic pattern is usually of fixed and known periods [4], for example, a pattern based on a 24 hour cycle. Further, a consumption time series may also demonstrate some variations due to the impact of some external factors, such as weather/temperature (extra demand for heating in winter and cooling in summer).

An overview of the proposed data generation method is provided in Figure 2. According to the figure, first, the periodic variations are adjusted from the actual time series. Next, a regression line is generated, which is extended for the forecasting purposes. In the end, periodic variations are added back to the newly generated time series in order to mirror reality. The reason to adjust periodic variations is due to the fact that the data with reduced periodic variations can produce more accurate forecasts than the data with variations [5]. Since, data with reduced periodic variations allows us to determine the best regression model for forecasting purposes. Reduced periodic fluctuations also assist in finding other components in the data, such as, trend, irregular variation and cycle.

B. Data Generation Model Design

In this section, we describe the process of generating synthetic time series data in detail, using a bottom-up approach. Algorithm 1, outlines the synthetic data generation process.

Algorithm 1 The process of synthetic data generation

```
1) Randomly select a smart meter time series as seed from
   the given cluster.
2) Adjust periodic variations in the selected time series by
   using a Ratio-to-Moving-Average method.
     a) The Ratio-to-Moving-Average method provides
        Periodic Indices (PIs) to reduce the periodic fluc-
        tuations.
3) Forecast the measurements (data points) for the periodi-
   cally adjusted time series by using Periodic Autoregres-
   sive (PAR) Models.
4) Add the periodic variations back to the forecast time
   series in order to simulate reality.
5) GO TO step 1, to generate further time series data sets.
```

According to Algorithm 1, first, an actual time series is randomly selected as 'seed' from the given cluster. Next, to

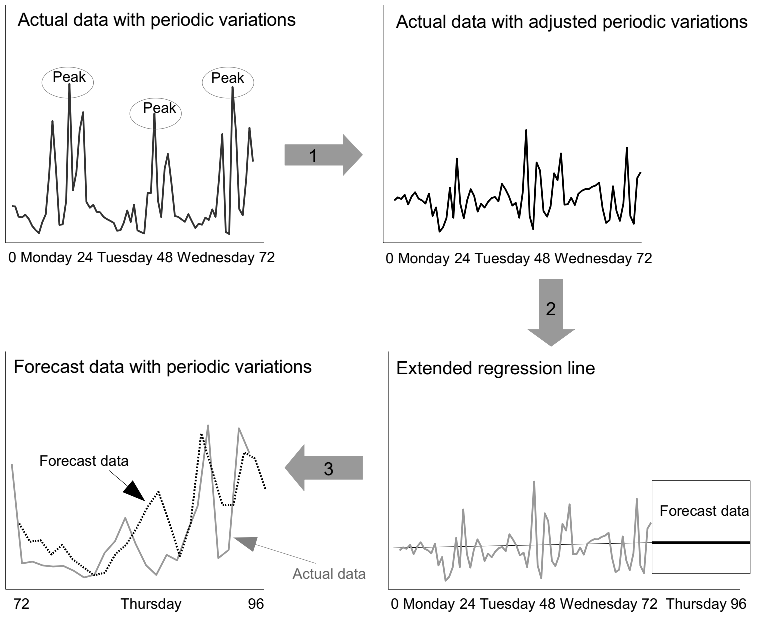

Actual data with periodic variations
Peak
Peak
Peak
0 Monday 24 Tuesday 48 Wednesday 72
1
Actual data with adjusted periodic variations
0 Monday 24 Tuesday 48 Wednesday 72
2
Extended regression line
Forecast data
0 Monday 24 Tuesday 48 Wednesday 72 Thursday 96
3
Forecast data with periodic variations
Forecast data
Actual data
72 Thursday 96

Fig. 2. An overview of the data generation process

reduce the impact of periodic fluctuations or to normalize the original time series, we calculate *Centered Moving Averages (CMAs)*. The *Centered Moving Average (CMA)* replaces the original time series with another series where the average of each point is centered at the middle of the data values being averaged [6]. The CMA of an even period (24-hour) is slightly more complex than the CMA of an odd period, however, it can be calculated as follows:

$$\begin{aligned} CMA\,(t) =& \tfrac{1}{2}\left(\frac{y_{t-12}+..+y_t+..+y_{t+11}}{24}\right) \\ &+\tfrac{1}{2}\left(\frac{y_{t-11}+..+y_t+..+y_{t+12}}{24}\right) \end{aligned} \tag{1}$$

where, $y_t, t = 1, ..., N$ is N consecutive observations of the time series. According to Equation 1, we include twelve readings on the left and eleven on the right of the observation, and eleven readings on the left and twelve terms on the right (that are offset by 1 hour relative to each other); as, neither of these readings are centered on at time t. If we now take the average of these two moving averages, we obtain a reading centered at time t. Further, Equation 2 describes the Raw Index (RI) or Ratio-to-Moving-Average. The RI is the ratio to the CMA, and it is computed by dividing the reading/data point by the CMA at time t. The RI is calculated for each data point in the time series.

$$RI(t) = \frac{y_t}{CMA(t)} \tag{2}$$

Based on the RI values, Equation 3 computes the Periodic Index (PI). As, we take an hour as a period. This corresponds to 24 computed periodic indices, each of which is the mean of the RI values at a particular hour in all days, for example, $PI(0)$ represents the mean of RI values at 0 o'clock in all days in the time series. In order to calculate the RI and the PI, we choose the multiplicative time series decomposition model [7], rather than the additive model. The multiplicative model is selected for the reason that it is more suitable when residuals are not of the same size [8], as in the smart meter investigated data sets.

$$PI\,(h) = \frac{1}{n}\sum_{i=0}^{n-1} RI(h+24i) \tag{3}$$

where, n represents the total number of days in the time series, and h is the hour of the day, i.e., 0 – 23. In addition, the base of the Periodic Indices (PIs) should be 1.00 [9], however, the way periodic indices are calculated there are chances of slight errors. For that reason, in Equation 4, PIs are normalized so that they sum to exactly 100% times the number of hours in a day (or 24 in our case) and their average should be 1.00.

$$PI'(h) = 24 * \frac{PI(h)}{\sum_{h=0}^{23} PI(h)} \tag{4}$$

Subsequently, Equation 5, computes the adjusted time series by dividing all the data points in the time series with the adjusted PIs. We use the resulting time series to generate the synthetic values by using the prediction method, described below.

$$y'_t = \frac{y_t}{PI'(h)} \tag{5}$$

where, $y'_t, t = 1, ..., N$ is N consecutive observations of the adjusted time series, $h = t \, mod \, 24$ and PI' is the normalized Periodic Index. The prediction method used in this paper is based on Periodic Autoregressive model (PAR) [2] and Normal distribution. In general, Auto-regression (AR) is a forecasting model that takes advantage of the relationship of values y_t to previous-period values $(y_{t-1}, .., y_{t-n})$. The AR is a multiple-regression model in which we try to predict a value of y from the previous values of y from previous time periods [10]. The regression coefficients (time-lagged versions of the dependent variables) in the AR models do not vary with the periods, whereas, in the PAR model the regression coefficients vary with the periods. For this reason, we need to calculate regression coefficients for each hour separately. In order to predict values using PAR models the value of p (period) is set to 24 (i.e., each hour of a day represent a period). For simplicity, we assume that $N/p = n$ is an integer. In other words, there are n full days of data available. The time index parameter, t, can be written $t = t(d, h)$, where $d = 0, ..., n-1$ and $h = 0, ..., p-1$. In this case, d and h denote the dth day and the hth hour in the day. The PAR model of order P can be written as follows, Equation 6.

$$\hat{y}'_{t(d,h)} = c + \sum_{i=1}^{P} \alpha_{i-1,h} y'_{t(d-i,h)} + \epsilon_{t(d,h)} \tag{6}$$

where c is the intercept with the $y-axis$ (a constant), α is the regression coefficient, ϵ is the value of the white noise and $d > p$. To generate data by prediction, we use the following PAR and Gaussian/Normal models, respectively (Figure 3). The PAR models are generated by the training process using the real-world data as seed.

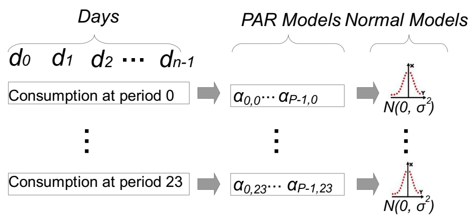


Fig. 3. The PAR and Normal Models

After the training process, we compute the estimated value $\hat{y}'_t$ at the time t by using the computed regression coefficients. Recall that we have reduced the period variance before generating the prediction model. We now "add" the variance back by multiplying it with the PI' and by adding the Gaussian white noise. Eventually, we generate the new value, y''_t, at the time t (Equation 7). Furthermore, the proposed data generation model is flexible enough to generate synthetic time series with different variances. In that case, the Periodic Indices $PI's$ could be from a separate time series within the same cluster hierarchy. This option allows us to generate many realistic time series with diverse variances.

$$y''_t = \hat{y}'_t * PI'(h) + \epsilon_t, \quad where \; h = t \, mod \, 24 \tag{7}$$

C. The Implementation

1) Data generation algorithm: The proposed data generator is designed to generate ample amounts of realistic data according to the given consumption patterns. We describe the implementation strategy of the data generator in Algorithm 2. The data generator takes *cluster*, *meters (number of meters)*, *days*, *start date* and *variance* as the input (line 1), and outputs the synthetic time series of energy consumption. The input cluster is the set of time series with somewhat similar consumption patterns, e.g., a cluster of private households - salaried employees. We generate different clusters using the *k-means clustering* algorithm [11] that divides all the time series into *k* segmentations based on the similarities in consumption patterns. Clustering is the pre-processing step in the data generation process, where a user can provide the k value to generate the desired number of clusters. The other four input parameters, *meters, days*, *starting date* and *variance* indicate the number of time series to generate, the length of each time series in days, the starting time and to generate time series with or without variance, respectively. The value of the variance can be '1' or '0'. Variance '1' means that the PIs are from any other time series, whereas, in case of '0' the PIs are from the same time series. Option '1' allows us to generate many realistic time series with diverse variances. For example, the function call of *SDATAGENERATOR('Privatehouseholdssalaried', 3, 365, '1-04-2016', 0)* represents that we would like to use cluster 'Private households salaried employees' as the seed to generate three time series, each of which contains the consumption readings of 365 days, starting from April 1st, 2016 and without variance. According to the algorithm, the data generator randomly picks a time series from the given cluster as seed (lines 2-3). Then, the generator computes average hourly energy values from the selected time series (line 4). The average value will be used as the default value in case the data generator predicts a negative value (lines 19-20). Further, the data generator creates a new time series with reduced periodic variations (lines 5-6). This is then used to train 24 PAR models, a separate PAR model is trained for each hour (line 7). If the value of the variance is '1', then the data generator randomly select another meter from the private households cluster and get or compute the $PI's$ (lines 8-10). Furthermore, for each hour of the day, the PAR model predicts a new value, i.e., it is a day-ahead prediction, producing 24 new hourly readings in total (lines 11-14). Each predicted value is multiplied by its corresponding PI' value or by another PI' value from a different time series (for reflecting the variance) (lines 15-18). The Gaussian white noise is also added to the reading (for simulating the effect of uncertainties in real world). In the end, the generated new reading is written to the output, along with meterID, the timestamp, and the granularity (hourly in this case) (lines 21-23). Based on the proposed data generation model design, it is also possible to aggregate time series data at higher levels of time granularity. The details of time series aggregation are discussed in the next section.

Algorithm 2 Smart Meter Data Generator

```
1: function SDataGenerator(Cluster cluster, Meters meters, Days days, StartDate startdate, Variance variance)
2:     for meterID ← 1, meters do
3:         m ← Randomly select a meter from the given cluster as seed/training data
4:         $\{average[0], .., average[23]\}$ ← Compute average energy values for 24 periods/hours
5:         $\{periodicindex[0,m], .., periodicindex[23,m]\}$ ← Compute and store normalized periodic indices for 24 periods of the selected meter
6:         $\left\{d[1]\left(\frac{y[0]}{periodicindex[0,m]}, .., \frac{y[23]}{periodicindex[23,m]}\right), .., d[n]\left(\frac{y[0]}{periodicindex[0,m]}, .., \frac{y[23]}{periodicindex[23,m]}\right)\right\}$ ← Reduce periodic fluctuations
                                                                      ▷ d represents day and y represents datapoint/meter reading
7:         $\{PAR[0], .., PAR[23]\}$ ← Compute PAR models using the seed
8:         if variance=1 then
9:             n ← Randomly select another meter from any (private households) cluster      ▷ Generate readings with different variance
10:            $\{periodicindex[0,n], .., periodicindex[23,n]\}$ ← get or (compute and store) normalized periodic indices for 24 periods
11:        for day ← 1, days do                                                          ▷ Generate synthetic data
12:            for period ← 0, 23 do
13:                $y[period]$ ← Generate the hourly reading using $PAR[period]$ model
14:                $\epsilon[period]$ ← Generate white noise using the Normal distribution function
15:                if variance=1 then
16:                    reading = $(y[period] * periodicindex[period,n]) + \epsilon[period]$   ▷ Multiple with the normalized periodic index of another meter
17:                else
18:                    reading = $(y[period] * periodicindex[period,m]) + \epsilon[period]$   ▷ Multiple with the normalized periodic index of the same meter
19:                if reading<0 then
20:                    reading = average($y[period]$)
21:                time ← Generate timestamp starting with the start date
22:                granularity ← 1                                                        ▷ hourly
23:                Write the output (meterID, time, reading, granularity)
```

Furthermore, to test and verify the anomaly detection algorithms in streaming data, the proposed data generator provides an optional feature to generate outliers or unusually high values. The anomaly is characterized as a series of unusual events, where an event is a consumer behavior that is reflected in the energy consumption pattern of the respective household [12]. The anomaly detection algorithms provide real-time notifications of these unusual events, such as leakage of energy. In addition, Equation 8, demonstrates the anomaly computation process. In Equation 8, v_p^d is the reading at period p and day d (Recall that each period represents an hour of the day, for that reason there are 24 periods in total). After that, v_p^d is compared against the average value of the readings $\bar{v}_p$ at period p of all days (Equation 9). Finally, based on the anomaly factor, the new amplified value $\hat{v}_p^d$ of v_p^d is calculated.

$$\hat{v}_p^d = \alpha * v_p^d \tag{8}$$

where

$$\alpha = \begin{cases} anomalyFactor & \textit{if } v_p^d > \bar{v}_p \\ \textit{1} & \textit{Otherwise} \end{cases} \tag{9}$$

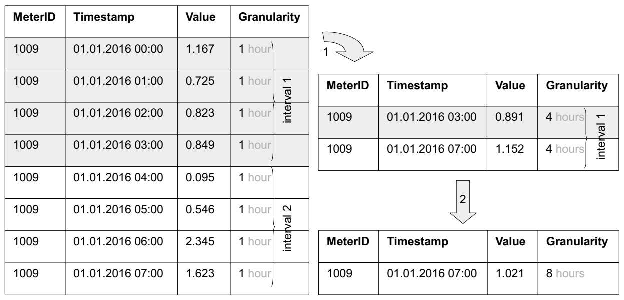
2) Generate Coarse-grained data: The proposed data generator has the ability to generate coarse-grained data. We use the *gradual granular aggregation* approach [13] to aggregate fine-grained data into coarse-grained data. The approach aggregates data at multiple levels of granularity. The reason to keep data at multiple levels is that users may want to apply various types of methods that analyze data at different levels of granularity. The data generator initially generates data at *hourly* level (meaning that each value is being recorded every hour), then it can be aggregated from an hour to higher levels, for example, 4 hour level, 8 hour level, day level and so forth.

The detailed data aggregation method is outlined in Algorithm 3. The aggregation method works as follows. It takes a *cluster*, *new granularity* and *data age* as input and outputs an aggregated time series (line 1). The new granularity (ng), represents the new level that the consumption readings should be aggregated to. For example, if the readings are at *hour* level, they can be aggregated to *4-hour* or even higher level. The data age (a), specifies that only the readings that are older than a specific time limit should be aggregated. For example, the function call, *SDATAAGGREGATION('Private households', 4, 3)* represents, aggregate all the time series in the cluster 'Private households' that are older than three months to *4-hour* level. For each row, the function checks if the reading is old enough to be aggregated. That is done by comparing the age of the reading with the required age (lines 5 & 8). The age of each value is computed by subtracting the current time stamp with the time stamp when the reading was generated. The lines 6-11 describe the aggregation process. The *previous time interval (pi)* and *time interval (i)*, calculates the beginning of each time interval based on the current granularity and aggregates several different levels of granularity into a single higher level. The extracted *hour* from the timestamp is first divided by *new granularity (ng)*, such as 4 (4 hour), 8 (8 hour)

Algorithm 3 Gradual aggregation

```
1: function SDataAggregation(Cluster c, NewGranularity ng, DataAge a)
2:     for all $ts \in c$ do                                          ▷ ts: time series
3:         m ← 0
4:         for all $datapoint \in ts$ do         ▷ datapoint: <pt.reading, pt.timestamp>
5:             if $(ca \geq a)$ AND $(datapoint = 1)$ then            ▷ ca: current age
                   ▷ ca = date difference in months (current.timestamp - pt.timestamp)
6:                 pi ← Get the first time interval, $\left\lfloor \frac{hour}{ng} \right\rfloor$        ▷ extracted hour
7:                 m ← pt.reading+m               ▷ add reading to the time interval
8:             if $(ca \geq a)$ AND $(datapoint > 1)$ then
9:                 i ← Get the next time interval, $\left\lfloor \frac{hour}{ng} \right\rfloor$
10:                if i=pi then
11:                    m ←pt.reading+m
                                          ▷ add reading to the previous time interval
12:                else
13:                    AVGreading ← Compute aggregated value, $AVG\left(\frac{m}{ng}\right)$
14:                    Write the output (ts, pt.timestamp, AVGreading, ng)
15:                    pi ← Get the value of time interval (i)
16:                    m ← 0
17:                    m ← pt.reading+m  ▷ add reading to the next time interval
```

and so on. Next, the *floor()* function is used to round the result to the greatest integer that is less than or equal to the value. For example, in Figure 4, there are eight hourly readings from 00:00 AM to 07:00 AM. Suppose, we would like to aggregate from 1 hour level to 4 hour level. In this case, each hour will be divided by the new granularity, i.e., 4. For that reason, hour 0 to hour 3 are divided by the new granularity value 4 and after rounding off the results, we get 0 that means that the first four values belong to the first time interval. Similarly, the next four values belong to the second time interval (lines 15-17) and so forth. The function aggregates the values of each time interval by using the AVG() function (line 13). Furthermore, it is possible to aggregate from 4 hour level to 8 hour level by dividing hour 3 and hour 7 with the new granularity, i.e., 8. Similarly, it is also possible to aggregate readings at higher levels.



MeterID	Timestamp	Value	Granularity
1009	01.01.2016 00:00	1.167	1 hour
1009	01.01.2016 01:00	0.725	1 hour
1009	01.01.2016 02:00	0.823	1 hour
1009	01.01.2016 03:00	0.849	1 hour
1009	01.01.2016 04:00	0.095	1 hour
1009	01.01.2016 05:00	0.546	1 hour
1009	01.01.2016 06:00	2.345	1 hour
1009	01.01.2016 07:00	1.623	1 hour

MeterID	Timestamp	Value	Granularity
1009	01.01.2016 03:00	0.891	4 hours
1009	01.01.2016 07:00	1.152	4 hours

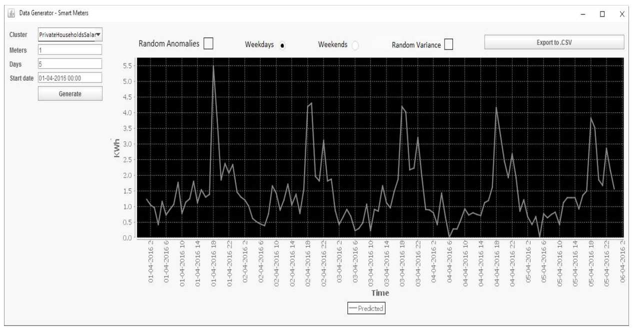
MeterID	Timestamp	Value	Granularity
1009	01.01.2016 07:00	1.021	8 hours

Fig. 4. Gradual granular aggregation of smart meter readings

3) User Interface: The implemented data generator supports data generation process through command-line and graphical user interface (Figure 5). A user can interact with the data generator by the GUI. First, the user needs to input the parameters, including the centroid of the cluster (shown as cluster), meters (number of meters), days (the length of the time series) and start date. If the user wants to add some unusual high values to test anomaly detection algorithms, he/she can simply check the *Random Anomalies* check-box. The user also needs to select the option of *weekdays* or *weekends*. The weekdays and weekends data is generated, separately. The user can also add variance by checking the *Random Variance* check-box. Afterwards, the user presses the *Generate* button to start generating the data. During data generation, the input parameters are sent to the data generation server through RESTful protocol in JSON format. The server generates the time series according to the given condition and send it back to the user. The time series is displayed in the graphic user interface where the user can explore the values by zoom-in and zoom-out features. The user can also click the *Export CSV* button to save the data locally.

IV. Evaluation

In this section, we evaluate the effectiveness and efficiency of the proposed data generator to generate the synthetic time series data. The data generator uses the real-world Irish electricity consumption data as 'seed' to generate synthetic data.



Cluster: PrivateHouseholdsSalar
Meters: 1
Days: 5
Start date: 01-04-2016 00:00
Random Anomalies ☐ Weekdays ● Weekends ○ Random Variance ☐

Fig. 5. GUI - Data generation tool

We evaluate the effectiveness of the proposed data generator by comparing the consumption patterns, distribution frequency and correlation between the real-world and synthetic data. We also evaluate the efficiency of the data generator by measuring the execution time. The experiments are conducted on a DELL Latitude E7440 laptop configured with an Intel Core i7-4600U Processor (2.1GHz, 4 Core, 8 Threads), 8 GB RAM and Samsung SSD driver (256GB). We run every test five times and compute the mean value after discarding the maximum and minimum values.

A. Effectiveness

We first compare the consumption patterns. The results are illustrated in Figure 6 (a-c) that demonstrate the differences between the synthetic and actual data on daily, weekly and six-monthly basis. For the daily consumption data (Figure 6 (a)), we can observe that the synthetic data matches the pattern of the actual data, for example, when the actual consumption increases from 3:00 PM to 23:00 PM, the consumption in the synthetic data also increases. Similarly, the weekly and six-monthly consumption data in Figure 6 (b and c), again demonstrates a pattern match between the synthetic and actual data. It is also observed that the actual data has more peaks than the synthetic data, illustrated in the six-monthly pattern (Figure 6 (c)). A user can actually obtain the desired peaks by adjusting the anomaly factor during the data generation process. Further, Figure 6 (d) compares the statistical information between the actual and synthetic data using histograms. The histograms show the frequency distribution of the values of the actual and synthetic data. A bar in the histogram represents the frequency/count of the occurrences of the values within an interval. According to Figure 6 (d), the frequency distribution of the synthetic data also indicates a notable fit to that of the actual data. Furthermore, Figure 6 (e), illustrates the correlation between the synthetic and actual data. The value of the correlation coefficient is *+0.7*. The coefficient value normally locates between -1 and +1. The positive value means that as one value increases the other also increases. The coefficient value of *+0.7* represents that *49%* of the variance between the two time series is related (*0.7 squared = 0.49*). Hence, this value indicates there exists a moderate correlation between the synthetic and actual data.

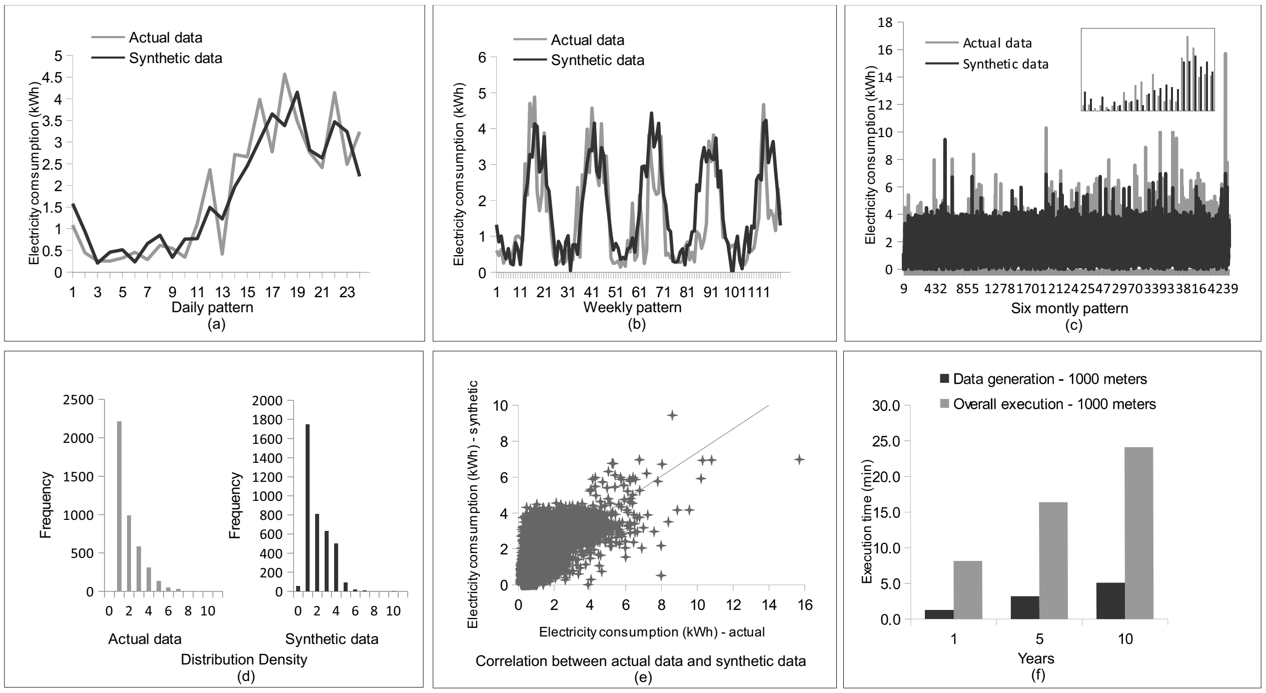


Fig. 6. Evaluation results

B. *Efficiency*

We now analyze the efficiency of the data generator by measuring the execution times. The data generation process consists of the following steps: 1) loading the input file (seed); 2) sorting and dividing the seed into weekdays and weekends/holidays; 3) computing twenty-four PAR models; 4) generating the synthetic data; and 5) writing the synthetic readings to an output file. We measure the overall execution time of these steps, and the time to generate data only (i.e., step 4). Figure 6 (f), shows the data generation time and overall execution time, respectively. According to the results, the data generator can achieve almost linear scalability to generate sizable data sets. It takes approximately 5 minutes to generate ten years data for 1,000 meters and 24 minutes for the overall execution. The results indicates that the data generator can generate data with a notably good performance.

To summarize, the proposed data generator has the ability to generate realistic time series data with reasonably good performance. The synthetic data has comparable characteristics with the actual data, with respect to the pattern and frequency distribution. The synthetic data demonstrates a reasonably good fit, however, a moderate correlation with the actual data.

V. Related Work

Synthetic data generation has been studied extensively across several disciplines. In the field of data management, *DBGEN* is a well-known data generation tool for use with the TPC-H benchmark [14]. DBGEN is often used to study systems performance. In the field of meteorology, synthetic weather data generation has also been extensively studied [15], [16], [17], [18], [19]. The weather generators typically use stochastic models to simulate synthetic weather data, such as wind, precipitation, rain, temperature and so on [15]. The stochastic models have the ability to maintain important statistical properties of the modeled data. Further, Markov chain models of first and second order are also used to generate arbitrary lengths of time series data [19]. Furthermore, a vehicle crash data generator has been proposed by Cuddihy et al. [20]. The data generator uses actual vehicle crash data as the seed to produce new realistic data using Fourier transformation. The generated data contains different acceleration peaks to test and verify crash management components in a car without running actual crash tests. Inspired by the vehicle crash data generator, we have also implemented an optional feature in the proposed data generator to generate accelerated peaks or outliers during the smart meter data generation process. These accelerated peaks or unusually high values can be used to develop, test and verify real-time anomaly detection algorithms. Real-time anomaly detection is one of the most important areas in smart meter data analytics that is under investigation by researchers.

Time series forecasting has also attracted much research attention in recent years. Zhang [21], proposed a hybrid time series forecasting model based on autoregressive integrated moving average (ARIMA) and neural networks. Anderson et al. [22], suggest periodic autoregressive moving average model (PARMA) for time series forecasting. PARMA model can explicitly describes seasonal/periodic fluctuations in terms of mean, standard deviation and autocorrelation. Based on that, PARMA derives more realistic time series forecasting models and simulations. Further, Kegel et al. [23] present Loom -

a template-based time series generation tool. Loom utilizes ARIMA as the underlying forecasting model. Additionally, Gooijer and Hyndman conducted a survey on forecasting models [24]. They have reported that ARIMA and neural networks are heavily used in time series forecasting. All the previous work has suggested that models such as stochastic, Markov chain, ARIMA, PARMA, neural networks play a crucial role in time series forecasting. In resemblance with these works, the foundation of the proposed data generator is based on PAR model. The underlying model ensures that the generated data simulate reality.

To the best of our knowledge, synthetic data generation in the field of smart meters has been much less studied as compared to the other disciplines. One of the reasons is that smart metering has been an emerging technology for some time. Nevertheless, work on Internet of Things (IoT) analytics benchmark [25] and benchmarking smart meter data analytics [26], have been found in the literature. The work by Arlitt et al. [25], uses Markov chain model, while the work by Liu et al. [26] uses a piece-wise linear regression model to generate synthetic smart meter data for bench marking purposes.

In contrast to all these works, our work mainly focuses on generating synthetic data that is based on the patterns of energy consumption. It provides detailed and concrete algorithms to generate smart meter data. It also suggests a practical mechanism of gradual granular aggregation of smart meter data.

VI. Conclusions and Future Works

Smart meter data analytics systems and algorithms require ample amounts of data for validation and benchmarking purposes. In this paper, we have presented a synthetic data generator to generate sizable volumes of realistic smart meter data. The proposed data generator generates data based on the energy consumption pattern using PAR model. We have also provided an easy-to-use GUI for interacting with the user. The GUI allows the users to generate synthetic data based on various input parameters, such as, centroid of the cluster, starting date, length of time series, weekdays or weekends data, variance and so on. We have evaluated the data generator by comparing consumptions patterns, distribution frequency and correlation between the actual and synthetic data. The results have demonstrated that the generator has the ability to generate synthetic consumption time series data that can simulate well to the real-world consumption time series data.

There are several directions for the future work. First, we would like to extend the proposed data generator by making it scalable, e.g., by using Hadoop MapReduce or Spark cluster to generate massive amounts of data. Second, we would also like to add more variables to customize the features of the generated time series, e.g., weather/temperature. Third, we plan to explore, how to store smart meter time series data in NoSQL databases. Fourth, we also want to generate other types of meter data, such as water, gas and heating. Last, but not least, we intend to investigate smart meter data cleansing.

Acknowledgements

This research is supported by UCN internal research and development funding (FOU-Project-6/2016) and the CITIES project funded by Danish Innovation Fund (1035-00027B).

References

[1] Smart Meter From Wikipedia.[Online]. Available: https://en.wikipedia.org/wiki/Smart_meter

[2] A.I. McLeod, "Diagnostic Checking of Periodic Autoregression Models with Application," *Journal of Time Series Analysis*, vol.15, no.2, pp.221–233, 1994.

[3] The Irish Social Science Data Archive (ISSDA).[Online]. Available: http://www.ucd.ie/issda/data/commissionforenergyregulationcer

[4] Time Series Components.[Online]. Available: https://www.otexts.org/fpp/6/1

[5] G.P. Zhang and M. Qi, "Neural Network Forecasting for Seasonal and Trend Time Series," *European Journal of Operational Research*, vol. 160, no. 2, pp.501–514, 2005.

[6] R. Weiers, *Introduction to Business Statistics*. Cengage Learning, 2010.

[7] G. Davis and B. Pecar, *Business Statistics using Excel*. Oxford University Press, 2013.

[8] Multiplicative Model.[Online]. Available: http://www-ist.massey.ac.nz/dstirlin/CAST/CAST/Hmultiplicative/multiplicative1.html

[9] K.D. Lawrence, R.K. Klimberg and S.M. Lawrence, *Fundamentals of Forecasting using Excel*. Industrial Press Inc., 2009.

[10] K. Black, *Business Statistics: For Contemporary Decision Making*. John Wiley & Sons, 2011.

[11] J. Wu, *Advances in K-means Clustering: A Data Mining Thinking*. Springer Science & Business Media, 2012.

[12] M. Jawurek, M. Johns and K. Rieck, "Smart Metering De-pseudonymization," *In Proc. of ACSAC*, pp.227–236, 2011.

[13] N. Iftikhar and T.B. Pedersen, "Using a Time Granularity Table for Gradual Granular Data Aggregation," *Fundamenta Informaticae*, vol.132, no.2, pp.153–176, 2014.

[14] M. Poess and C. Floyd, "New TPC Benchmarks for Decision Support and Web Commerce," *SIGMOD Record*, rec.29, no.4, pp.64–71, 2000.

[15] K. Breinl, T. Turkington and M. Stowasser, "Simulating Daily Precipitation and Temperature: A Weather Generation Framework for Assessing Hydrometeorological Hazards," *Meteorological Applications*, vol.22, no.3, pp.334–347, 2014.

[16] Li.Z.F. Brissette and J. Chen, "Finding the Most Appropriate Precipitation Probability Distribution for Stochastic Weather Generation and Hydrological Modeling in Nordic Watersheds," *Hydrological Processes*, vol.27, no.25, pp.3718–3729, 2013.

[17] K. Breinl, T. Turkington and M. Stowasser, "A Weather Generator for Hydro-meteorological Hazard Applications EGU General Assembly Conference," *In EGU General Assembly Conference Abstracts*, vol.16, pp.10522, 2014.

[18] A.H. van Paassen and Q.X. Luo, "Weather Data Generator to Study Climate Change on Buildings," *Building Services Engineering Research and Technology*, vol. 23, no. 4, pp.251–258, 2002.

[19] A. Shamshad, M.A. Bawadi, W.W. Hussin, T.A. Majid and S.A.M. Sanusi, "First and Second Order Markov Chain Models for Synthetic Generation of Wind Speed Time Series," *Energy*, vol.30, no.5, pp.693–708, 2005.

[20] M.A. Cuddihy, J.B. Drummond Jr and D.J. Bourquin, "Ford Motor Company, Vehicle Crash Data Generator," U.S. Patent 5,608,629, 1997.

[21] G.P. Zhang, "Time Series Forecasting using a Hybrid ARIMA and Neural Network Model," *Neurocomputing*, 50, pp.159–175, 2003.

[22] P.L. Anderson, M.M. Meerschaert and K. Zhang, "Forecasting with Prediction Intervals for Periodic Autoregressive Moving Average Models," *Journal of Time Series Analysis*, vol. 34, no.2, pp.187–193, 2013.

[23] L. Kegel, M. Hahmann and W. Lehner, "Template-based Time Series Generation with Loom," *In EDBT/ICDT Workshops*, 1558, 2016.

[24] J.G.D. Gooijer and R.J. Hyndman, "25 Years of Time Series Forecasting," *International Journal of Forecasting*, vol.22, no.3, pp.443–473, 2006.

[25] M. Arlitt, M. Marwah, G. Bellala, A. Shah, J. Healey and B. Vandiver, "IoTA Bench: An Internet of Things Analytics Benchmark," *HP Laboratories Technical Report*, HPL-2014-75.

[26] X. Liu, L. Golab, W. Golab and I.F. Ilyas, "Benchmarking Smart Meter Data Analytics," *In Proc. of EDBT*, pp.385–396, 2015.