Scientia Iranica D (2017) 24(6), 3087–3100

Sharif University of Technology
Scientia Iranica
Transactions D: Computer Science & Engineering and Electrical Engineering
www.scientiairanica.com

Comparison and analysis of three anomaly correction methods in embedded systems

R. Mojarad* **and H.R. Zarandi**

Department of Computer Engineering and Information Technology, Amirkabir University of Technology, 424 Hafez Ave, Tehran, P.O. Box 15875-4413, Iran.

Received 28 December 2014; received in revised form 8 December 2016; accepted 11 March 2017

KEYWORDS
Anomaly;
Anomaly detection;
Anomaly correction;
Correction coverage;
Embedded systems.

Abstract. This paper proposes and compares three anomaly correction methods in embedded systems: 1) Markov-based; 2) Stide-based (sequence time-delay embedding); 3) Cluster-based correction methods. All these methods work online on data streams coming from sensors of embedded systems. In these methods, detection is first obtained using training on normal data, and next in runtime, the correction mechanisms can be applied. Markov-based method is based on probabilities between states, Stide-based method is based on storing common events, and Cluster-based one is based on clustering similar members. In the detection phase, these methods check normal behavior of input data based on what is learned at the training phase. Evaluation is performed using 7000 data sets. The window size and number of injected anomalies vary between 3 and 5, 1 and 7, respectively. Correction coverage for Markov-based, Stide-based, and Cluster-based methods is on average 77.66%, 60.9%, and 70.36%, respectively. Therefore, Markov-based method is the best in terms of correction coverage. Moreover, area overheads of these methods are 249.64, 63.35, and 2.08 μm^2, respectively. As a trade-off, Cluster-based method shows the best correction coverage compared to area, power, and delay overheads.

1. Introduction

Embedded systems are widely used in modern critical digital systems design. Their advantages over superscalar processor architectures are [1,2]: low power consumption, high performance, and low design complexity. The spread pace of smart systems calls out for a focus on embedded systems, defined as a controlling and management unit embodied in a larger system [1,2]. Modern vehicles, home appliance, and medical instrument are mere examples of the vast application of embedded systems [3]. Considering this spread, it is vital for them to be fault-tolerant in order to prevent the resulting damages.

These systems typically consist of three components: sensor, processing unit, and actuator. Sensors play a significant role in the operation of embedded systems [4]. Therefore, the fault in their data, so-called anomaly, is important to consider. There are two types of methods for tolerating faults in this regard [5]: anomaly detection and anomaly correction. Detection of anomaly analogous to fault detection can be done explicitly or implicitly; in the former, the detection is based on looking for signs of specific type of faults [6,7]. However, in the latter, the behavior of data as indirect indicators is used for detection. As an example, an Ethernet broadcast storm is marked indirectly by high packet traffic on a network abnormally [8]. Here, packet traffic is measured by sensors.

The significance of detection and correction meth-

*. *Corresponding author. Tel.: +98 21 64545119*
E-mail addresses: roghaye_mojarrad@aut.ac.ir (R. Mojarad); h_zarandi@aut.ac.ir (H.R. Zarandi)

doi: 10.24200/sci.2017.4579

ods directly depends on the value of data [9-11]. The data generated in sensors, also known as sensor data, can be numerical or categorical [8]. The former behave like numbers, which are scalable, continuous, and have a zero, on which the mathematical operations can be performed. However, the latter are discrete without any order or mathematical operations [8]. Categorical data are presented as a string of symbols; hence, their anomaly could result from unknown symbol(s) or unknown sequence(s) of symbols [8]. It should be mentioned that the popularity of sensors with categorical output is increasing due to enhancement of computational power [8]. Due to its nature, statistical analysis cannot be used, which makes the anomaly detection and correction much more challenging. The anomaly correction plays a vital role in the reliability, safety, and generally dependability of embedded systems.

While there have been some studies on anomaly detection, not so many studies have been done on the anomaly correction.

This paper is the extended version of the work presented in [12] as follows:

- Another correction method, Cluster-based one [13], is also added to this work to be compared with two other methods, i.e. Markov-based [14] and Stide-based ones [12], to get better considerations of correction methods;
- Detection and correction algorithms of Cluster-based method are presented in detail;
- Correction algorithm of Markov-based method is presented in detail;
- Comparisons between results achieved by three different methods are done and their advantages/disadvantages are discussed;
- The best trade-off between correction coverage versus area, power and delay overhead is introduced and the best method is specified.

In this paper, three anomaly correction methods are introduced:

1. Markov-based;
2. Stide-based;
3. Clustering-based methods.

They all consist of three phases: training, anomaly detection, and correcting the detected anomaly. In the training phase, the anomaly correction system is set up to distinguish between the normal and anomalous data. Therefore, in Markov-based, Stide-based, and clustering-based methods, a transition matrix, list of normal sequences along with their frequencies, and clusters of similar sequences are generated, respectively. In order to correct an anomaly, it is necessary to detect it beforehand, and this will be done in anomaly detection phase. It is done based on the output of the training phase in all the methods. Eventually, the system tries to correct the detected anomaly based on the models generated in the training phase. In this phase, the system tries to correct the data by unique substitution, multiple substitution or deletion.

These methods only see and operate within a specified portion of data, called window size. The number of constraints and meta-data directly corresponds to the length of this window. Another term used in this paper is threshold, which is the maximum acceptable distance from normal data without being considered as an anomaly.

In order to evaluate the proposed methods, it is required to calculate the correction coverage, power and area consumption in addition to delay. This was done by 7000 datasets ready for test with various numbers of anomalies. The number of injected anomalies ranged from 1 to 7. The results were in favor of the performance of the methods. Correction coverages for Markov-based, Stide-based, and Cluster-based methods are on average 77.66%, 60.9%, and 70.36%, respectively. Therefore, Markov-based method is the best in terms of correction coverage. Moreover, area overheads of these methods are 249.64, 63.35, and 2.08 μm^2, respectively. With the help of introducing a metric that considers correction coverage and overheads, Cluster-based method is proposed as the best one in which fault-tolerant property is traded off against overheads.

The rest of the paper comes in the following order: In Section 2, the background knowledge is discussed, while the previous studies and their details of the proposed methods are presented in Sections 3 and 4, respectively. Moreover, Section 5 is depicted to evaluate methods and their results. The paper ends with conclusion in Section 6.

2. Background knowledge

It is necessary to provide the definition of the keywords in this context, some of which are in the following [8]:

- *Surprise factor*: Defined as the inverse probability of an event; in other words, its possibility of not happening;
- *Categorical data:* The types of data which are not modelled as numbers whilst there are no mathematical relations between them;
- *Training dataset:* Defined as the dataset used for preparing the system for working with real-world data;
- *Testing dataset:* The dataset which is the samples of real-world data and used to estimate the behavior of the system in real world;

- *Normal dataset:* This dataset is obtained from training dataset and includes normal data without any anomaly.

2.1. Anomalous event

Anomalies are grouped into: unknown symbol, unknown sequence(s), and rare sequence(s) which are defined as follows [8]:

1. *Unknown symbol:* If a symbol occurs in the data unprecedentedly in the testing phase, then it is flagged as an unknown symbol;
2. *Unknown sequence:* This is the generalized form of an unknown symbol if a set of symbols that constructs a sequence is unprecedented in the training phase, even if all its symbols are normal;
3. *Rare sequence:* This type of sequence is similar to an unknown sequence with only considering the frequent sequences in the training phase; in other words, if a sequence occurred in the training phase, but its number of occurrence is less than the rare-threshold, it would be considered as a rare sequence. The rare threshold is the minimum number of occurrence in which the sentence is considered as a normal sequence. It is worth mentioning that the threshold of occurrence number needs to be determined.

2.2. Anomaly detection

Anomaly detection utilizes a portion of the data stream for its detection, and the size of this portion is known as a window size [8]. Anomaly detection has a window size [8] which determines the length seen from data stream. Anomaly detection can be analyzed only within this scope.

3. Related work

There have been several studies on anomaly detection [15-17]; nevertheless, they are not capable of detecting anomalies in categorical data like TMR (Triple Modular Redundancy) and DWC (Duplication With Comparison); moreover, some studies have this ability only for specific types of categorical data such as eye-detection sensors [18]. Markov [8], Stide [8], probability-based [19] and buffer-based methods [19] are some of the most significant anomaly detection methods presented for categorical data. Since Markov-based and Stide-based detection methods are utilized for generating two proposed methods, they are presented in a more detailed fashion as follows.

3.1. Markov-based anomaly detection [8]

Markov-based anomaly detection method consists of three steps:

1. *Training*, in which all of the transitions are analyzed and their respective probabilities are calculated. This information forms a transition matrix;
2. *Threshold setting*, where a suitable threshold is determined based on the transition matrix;
3. *Testing*, in which the probability of each transition in the testing dataset is calculated and compared with the threshold. Not surprisingly, when the complementary probability of the transition exceeds the threshold, the system will flag this transition as anomaly.

This method is able to detect anomaly in the stream of data by modeling the data into Markov model. The key feature of this model is that each state depends only on its previous state [8,20]. This feature is depicted in Eq. (1), in which $X_{(t+1)}$ and $X_{(t)}$ are the next and current states, whilst the rest are all previous states from initial state X_0 [8]:

$$P(X_{t+1}=x_{t+1}|X_t=x_t, X_{t-1}=x_{t-1}, \cdots, X_0=x_0) = P(X_{t+1}=x_{t+1}|X_t=x_t). \quad (1)$$

Transition matrix is generated during the training step by analyzing and calculating the probability of each transition between two states. That is done by counting the number of all transitions initiated from state A and those terminated in B, along with the division of the latter by the former to evaluate the probability of transition $A \rightarrow B$. This calculation is shown in Eq. (2), in which $F(A,B)$ is the number of times when B comes after A, and $F(A)$ is the total number of transitions after A:

$$P(A,B) = \frac{F(A,B)}{F(A)}. \quad (2)$$

The threshold value is empirically determined based on the value in the matrix, and their complement is a surprise factor of the events.

3.2. Stide-based anomaly detection [8]

Stide-based anomaly detection similarly utilizes three steps: Training datasets will be divided into overlapping sequences with a defined length of N, whose value is determined in practice [21]. After removing duplicates from these sequences, they will be stored in a database along with their frequencies. Furthermore, the testing dataset is similarly divided into overlapping sequences with equal length of N. The detection system looks up for each sequence in the database, and if the search was successful, its score would be zero, and one otherwise. The frame with the maximum number of ones would show the place of the anomaly [20].

3.3. Probability-based anomaly detection [19]

This method utilizes the relative distance of symbols. Likewise, it consists of three steps. In the training step, the probability matrix is generated, in which rows are possible permutations of symbols in the dataset and columns are possible distance of two symbols in one pair. In other words, element $E_{i,j}$, which is placed in the ith column and jth row, is equal to probability of the jth pair occurrence with exactly i symbol between them. After formation of probability matrix, the probability multiplication function is used to determine the normality of each sequence. Then, if the value exceeds the threshold, it will be detected as anomaly [19].

3.4. Buffer-based anomaly detection [19]

This method is analogous to Stide-based one with slight alteration. That is, in the training step, all unique sequences are obtained, and in the end of this step, the rare sequences from the database are removed. In the testing step, if the search for the sequences in the database was successful, the score would be zero and one if otherwise; afterwards, if the unsuccessful searches exceed the defined threshold, it will be marked as anomaly [19].

4. Three correction methods: Markov-based, Stide-based, and cluster-based methods

All methods proposed in this paper operate in three steps:

1. Training;
2. Anomaly detection;
3. Anomaly correction.

The first two steps are analogous to their respective steps in anomaly detection method, discussed in the last section. In order to correct the anomaly successfully, first, the anomaly sequence is identified, and afterwards all its symbols in this window size are analyzed. It is noteworthy that those systems with larger window size take in more information and can detect and correct better; however, it will increase the overheads of the system; hence, a suitable anomaly detection or correction method should be able to work favorably with small-sized window. For correction, the system considers all possible options of substitution and deletion of the symbols, and the most suitable one will be chosen.

4.1. Anomaly correction

To correct an anomalous event, more than one state may occur: *unique substitution*, *multiple substitutions*, and *deletion*.

- *Unique substitution:* If only one of the options meets the constraints;
- *Multiple substitutions:* If there is more than one option to meet the requirement, the frequency of their corresponding transition is calculated and using weighted random selection based on their frequency, the most probable corrected event is selected;
- *Deletion:* If none of substitution options works, the systems tries to find the corrected event by considering the elimination of each symbol.

4.2. Markov-based anomaly correction

In Markov-based method, the number of constraints is in direct relationship with the size of the window, that is, for the system with window sizes equal to 3 and 4, the number of constraints would be 12 and 20. To illustrate that, assume that the training data is *KLMNOPABH*, testing sequence is equal to *KLMNOLABH*, and window size is set to 3. In the training step, the transition matrix is formed. Figure 1 shows transition matrix of this particular example.

If the threshold value is equal to 0.9, the anomaly detection is performed in the following way:

$$\begin{aligned} &P(KLM, LMN) = 1 \\ &\Rightarrow \text{Surprise Factor} = 1 - P(KLM, LMN) \\ &\qquad = 0 < 0.9 \qquad \checkmark \\ &P(LMN, MNO) = 1 \\ &\Rightarrow \text{Surprise Factor} = 1 - P(LMN, MNO) \\ &\qquad = 0 < 0.9 \qquad \checkmark \\ &P(MNO, NOL) = 0 \\ &\Rightarrow \text{Surprise Factor} = 1 - P(MNO, NOL) \\ &\qquad = 1 > 0.9 \qquad \times \end{aligned} \tag{3}$$

Based on Eq. (3), the third transition is not normal and the system correctly detects that as anomaly. As mentioned, the correction system first needs to identify the location of the anomaly. Table 1 shows the required constraints for locating the place of

	KLM	*LMN*	*MNO*	*NOP*	*OPA*	*PAB*	*ABH*
KLM	0	1	0	0	0	0	0
LMN	0	0	1	0	0	0	0
MNO	0	0	0	1	0	0	0
NOP	0	0	0	0	1	0	0
OPA	0	0	0	0	0	1	0
PAB	0	0	0	0	0	0	1

Figure 1. Transition matrix for mentioned example in Markov-based method.

Table 1. Constraints for state of substitution of the example in Markov-based method.

First symbol	Second symbol	Third symbol
$KLM \rightarrow LM?$	$LMN \rightarrow MN?$	$MNO \rightarrow NO?$
$LM? \rightarrow M?O$	$MN? \rightarrow N?L$	$NO? \rightarrow O?A$
$M?O \rightarrow ?OL$	$N?L \rightarrow ?LA$	$O?A \rightarrow ?AB$
$?OL \rightarrow OLA$	$?LA \rightarrow LAB$	$?AB \rightarrow ABH$

? shows that the symbol can be any symbol.

anomaly, where the question mark is for the unknown placeholder and an arrow is used to show a transition; in other words, $KLM \rightarrow LM?$ conveys that the system looks up the possible transitions initiating from KLM to all the states with L and M as their two first entities. An option filling the question mark is a suitable one, if and only if all the question marks in one group of the constraints could be replaced by that option and all of the transitions would be normal. In the particular example, the first two symbols cannot be substituted with any symbols to meet the constraints, and the details are shown in Table 2.

Contrary to the first two symbols, the third symbol is the location of anomaly. Because all of the corresponding constraints are met, the system, therefore, searches for an alternative to substitute it with the question mark. In this example, the system chooses P because it meets all of the constraints, and because no other symbol is able to fit in them; that would be a unique substitution. If the correction system cannot find the suitable substitution to correct the anomalous event, the deletion case should be considered. Table 3 depicts the constraints of deletion case. For better assurance, the detector does not move forward after correction and reruns on the current window.

In the example, only one anomaly is in the window, and it is not always the case. When there are multiple anomalies in a window, the system uses

Table 2. States of constraints meeting for the first and second symbols of anomalous sequence example in Markov-based method.

First symbol		Second symbol	
$KLM \rightarrow LM?$	OK	$LMN \rightarrow MN?$	OK
$LM? \rightarrow M?O$	OK	$MN? \rightarrow N?L$	Not-OK
$M?O \rightarrow ?OL$	Not-OK	$N?L \rightarrow ?LA$	Not-OK
$?OL \rightarrow OLA$	Not-OK	$?LA \rightarrow LAB$	Not-OK

similarity function to find the most similar sequence to anomalous one, such that if it replaces the anomalous event, it would be considered as normal. The system uses a basic similarity function named as Overlap [8] which is shown in Eq. (4), and the most similar one would be considered as the correct form. The pseudocode of the Markov-based anomaly correction is presented in Figure 2:

$$S_k = \begin{cases} 1 & \text{if } (X_k = Y_k) \\ 0 & \text{Otherwise} \end{cases} \Rightarrow \text{sim}(X, Y) = \sum_{k=0}^{N} S_k. \quad (4)$$

In the pseudocodes, W shows window size of correction method, and variable *Detect-Anomaly* is a flag to show the occurrence of anomaly. Variable *CorrectionState* is obtained by checking the constraints and it determines if *Unique Substitution, Multiple Substitution*, or *Deletion* can correct the anomaly. If CorrectionState equals Deletion, it means that neither Unique Substitution nor Multiple Substitution can correct the anomaly; and deletion must be used. Moreover, *NumberofDeletion-State* depicts the number of candidates for correction by deletion, and the anomaly will be corrected by deleting different symbols of anomalous event, and if this value is equal to zero, then the similarity function is utilized to correct the anomaly.

4.3. Stide-based anomaly correction

Analogous to Markov-based anomaly correction, Stide-based one works in three phases with similar behavior. In the training phase of the method, as mentioned, anomaly detection method forms a database of unique normal sequences with their frequency. Moreover, the threshold value is heuristically determined based on the database. In the anomaly detection phase, the method compares the frequency of the occurred sequence in the database with the threshold, and if it exceeds the occurred sequence, it will be marked as anomaly and directed to the correction phase.

In order to locate the position of anomaly in Stide-based anomaly correction, similar to Markov-based one, the constraints should be checked. That is, for example, if the window size is equal to three, nine constraints should be verified. It is worth mentioning that in contrast to Markov-based anomaly correction, Stide-based anomaly correction method only analyzes the current sequence, not the transitions. When the source of anomaly is identified, similar to Markov-based correction method, the system tries to substitute a

Table 3. Constraints of deletion state for the example in Markov-based method.

First symbol		Second symbol		Third symbol	
$KLM \rightarrow LMO$	Not-OK	$LMN \rightarrow MNL$	Not-OK	$MNO \rightarrow NOA$	Not-OK
$LMO \rightarrow MOL$	Not-OK	$MNL \rightarrow NLA$	Not-OK	$NOA \rightarrow OAB$	Not-OK
$MOL \rightarrow OLA$	Not-OK	$NLA \rightarrow LAB$	Not-OK	$OAB \rightarrow ABH$	Not-OK

```
Input: Training Data, W, Testing Data
Output: Corrected Data

/*Training Step: Build Transition Matrix*/
1. Find all states (sequence with size W) and transitions of Training Data
2. Calculate probability of transitions from one state to another state based on Training Data
3. Build Transition Matrix using calculated probabilities

4. FOR ALL TestingSequence IN Testing Data
      /*Detection Step*/
5.      SurpriseFactor = Obtain the probability of each testing transition using Transition Matrix
6.      IF SurpriseFactor > Threshold THEN
7.          DetectAnomaly = TRUE
8.          Anomalous Event = TestingSequence
9.      ELSE
10.         DetectAnomaly = FALSE
11.         Move Window Forward
12.     END IF

        /*Correction Step*/

13.         IF DetectAnomaly = TRUE THEN
14.             Find Source of Anomaly from Anomalous Event
15.             CorrectionState = Consideration which state can correct that anomaly
16.             IF CorrectionState = Unique Substitution THEN
17.                 Correct Training Data using symbol replacement
18.             ELSE IF CorrectionState = Multiple Substitution THEN
19.                 Select one of possible correction state using weighted probabilities
20.             ELSE IF CorrectionState = Deletion THEN
21.                 NumberofDeletionState = Find number of deleting states needed for correction
22.                 IF NumberofDeletionState = 1 THEN
23.                     Correct Training Data using delete operation
24.                 ELSE
25.                     Correct Training Data based on one of possible correction state using on their probabilities
26.                 END IF
27.             ELSE
28.                 Correct Training Data using similarity function to find best state for anomalous event
29.             END IF
30.         END IF
31.END FOR

32. RETURN Corrected Testing Data
```

Figure 2. The Markov-based anomaly correction algorithm.

single and multiple symbols or delete the symbols. In *Multiple Substitutions*, which are more than one state, the anomaly can be corrected, so the weighted function is used to select one of those states.

To illustrate how the method works, assume that the training data are the same as those of the previous example, for which the formed database is shown in Table 4. With a similar testing dataset, assuming that the threshold equals zero, the detector will identify the fourth sequence as anomaly, as the previous three are presented in the database. In order to locate the anomaly, the constraints are generated for the anomalous sequence; as for this example, it is rather straightforward and presented in Table 5. Analyzing the constraints shows that the third symbol in the anomalous sequence is the source of anomaly because the required constraints can be met. In this example, unique substitution will correct the anomaly, and it is not required to consider multiple substitutions and deletion. Symbol 'P' is a suitable choice for correction.

Table 4. Database of normal sequences as an example in the Stide-based correction method.

Sequence	Frequency	Sequence	Frequency
KLM	1	OPA	1
LMN	1	PAB	1
MNO	1	ABH	1
NOP	1		

Table 5. States of constraints meeting for symbols of anomalous sequence example in the Stide-based correction method.

First symbol		Second symbol	
$LM?$	OK	$MN?$	OK
$M?O$	OK	$N?L$	Not-OK
$?OL$	Not-OK	$?LA$	Not-OK

Table 6. Constraints for state of deletion for the example in Stide-based method.

First symbol		Second symbol		Third symbol	
LMO	Not-OK	MNL	Not-OK	NOA	Not-OK
MOL	Not-OK	NLA	Not-OK	OAB	Not-OK
OLA	Not-OK	LAB	Not-OK	ABH	OK

Table 6 shows the required constraints to analyze the *Deletion* state in order to correct the anomalous event.

4.4. Cluster-based anomaly correction

Similar to Stide-based method, in the training phase, the normal data are broken into overlapping sequences of fixed length, W, denoted as the size of the window. These sequences are used for making clusters of data. For that purpose, a *membership threshold* is determined, which is the minimum similarity required between members of a cluster. The comparison between sentences should be done using similarity functions, which can be determined based on the types of data in each specific application. In this context, two sequences are similar if and only if the yielding result of their corresponding similarity function is a value greater than the membership threshold.

The system is trained in the following way. After splitting the normal data into fixed length sequences, they will be processed and clustered together. If a sequence is similar to all sequences in a cluster, it would be added to the cluster; if there is no cluster, for which the sequence meets the criteria, a new cluster will be created with only one member. At the end of training phase, a representative is selected randomly for each cluster.

In the detection phase, the testing data will be broken into fixed length sequences, and each sequence is compared with all the representatives; if there is no representative for it, the similarity function yields a greater or equal value rather than *anomaly threshold*, and the sequence will be considered as anomaly. Otherwise, the members of cluster(s) whose representatives are similar to the sequence will be compared to the sequence. In other words, a sequence is flagged as normal when the system finds one cluster whose all members are similar to that sequence. The system checks similarity with representatives before verifying the similarity value for all the other members.

In the correction phase, the most similar representative replaces the anomalous sequence. To illustrate that, assume that *ABCABFBKALCKBA* is the normal data in the training phase, and window size and membership threshold are determined as 3 and 2, respectively. The details of forming the clusters based on the training data and *Overlap* similarity function are depicted in Table 7, where for each sequence, the conditions are checked and if there is a suitable cluster to join, the action will be to *join* and *create* if otherwise. Eventually, nine clusters are constructed in the training phase and one representative for each cluster. During the testing phase, assume that the testing data are *ABCLBFBK*, and the anomaly threshold value is two.

1. $ABC \rightarrow$ Cluster 1, (sim(ABC, ABC) $= 3 > 2$);
2. BCL $\rightarrow$ Anomaly, since there is no similar representative for it.

In order to make correction, the sequence is substituted with the most similar sequence. Table 7 shows the similarity between the anomalous sequence and all other sequences, and clearly sequence BCA is the most similar choice and, therefore, is selected for correction. The pseudocode of Cluster-based anomaly correction is presented in Figures 3 and 4. Various similarity functions exist which can be used in this method [22]: *Overlap*, *Skin*, *Goodall1*, *Goodall2*, *Goodall3*, *Goodall4*, *Gambaryan*, and *Burnaby*. These are implemented and evaluated for comparison in the context of the anomaly correction.

5. Experimental study

The embedded system receives and learns about the outside world via sensors, and that is where anomaly happens. Hence, the anomaly correction should be located just after the sensor. As stated in the previous sections, the proposed methods of correcting an event need to investigate events in the range of window. Although it will insert some delay into data, it can work in a real-time system. A real-time system is defined as a system which can provide suitable response in less than a specified delay. Since this system can work with the delay of the half of window events in addition to an overhead of calculation, the proposed anomaly correction methods can be considered as real-time systems.

The normal datasets are extracted from the University of New Mexico's website [23], which are synthetic data for *sendmail* functions at the aforementioned university on Sun SPARC stations running unpatched 4.1.1 and 4.1.4 [23]. The three proposed methods are evaluated in three steps:

1. To extract the testing dataset, a utility program is developed. It takes a random part of the normal

Table 7. Constructing of clusters in the proposed method.

Sequence	Reasoning	Action	Clusters
ABC	Since there is no cluster	Create	{**ABC**}
BCA	$Sim(ABC, BCA) = 0 < 2$	Create	{ABC}, {**BCA**}
CAB	$Sim(CAB, ABC) = 0 < 2$; $Sim(CAB, BCA) = 0 < 2$	Create	{ABC}, {BCA}, {**CAB**}
ABF	$Sim(ABF, ABC) = 2 \not< 2$	Join	{ABC, **ABF**}, {BCA}, {CAB}
BFB	$Sim(BFB, ABC) = 0 < 2$; $Sim(BFB, BCA) = 1 < 2$ $Sim(BFB, CAB) = 1 < 2$	Create	{ABC, ABF}, {BCA}, {CAB}, {**BFB**}
FBK	$Sim(FBK, ABC) = 1 < 2$; $Sim(FBK, BCA) = 0 < 2$; $Sim(FBK, CAB) = 0 < 2$; $Sim(FBK, BFB) = 0 < 2$	Create	{ABC, ABF}, {BCA}, {CAB}, {BFB}, {**FBK**}
BKA	$Sim(BKA, BCA) = 2 \not< 2$	Join	{ABC, ABF}, {BCA, **BKA**}, {CAB}, {BFB}, {FBK}
KAL	*Not similar to any cluster*	Create	{ABC, ABF}, {BCA, BKA}, {CAB}, {BFB}, {FBK}, {**KAL**}
ALC	*Not similar to any cluster*	Create	{ABC, ABF}, {BCA, BKA}, {CAB}, {BFB}, {FBK}, {KAL}, {**ALC**}
LCK	*Not similar to any cluster*	Create	{ABC, ABF}, {BCA, BKA}, {CAB}, {BFB}, {FBK}, {KAL}, {ALC}, {**LCK**}
CKB	$Sim(CKB, CAB) = 2 \not< 2$	Join	{ABC, ABF}, {BCA, BKA}, {CAB, **CKB**}, {BFB}, {FBK}, {KAL}, {ALC}, {LCK}
KBA	*Not similar to any cluster*	Create	{ABC, ABF}, {BCA, BKA}, {CAB, CKB}, {BFB}, {FBK}, {KAL}, {ALC}, {LCK}, {**KBA**}
Select representative (marked as underline)			{ABC, ABF}, {BCA, BKA}, {CAB, CKB}, {BFB}, {FBK}, {KAL}, {ALC}, {LCK}, {KBA}

dataset as the background data and stored as normal data; afterwards, considering [23], the program randomly injects anomalies into the normal data to be stored as testing data. This program inserts one to seven anomalies and generate 1000 testing data for each number of injected anomalies;

2. The proposed methods are implemented using VHDL. The key factor in terms of fault-tolerant embedded system is correction coverage. For its calculation, the proposed methods runs over 7000 testing datasets and corrects them. The system is synthesized using Synopsis Design Compiler with 45 nm *Nangate opencell* library for hardware analysis that is estimating power, area and time consumption [24,25];
3. For evaluating the correction coverage, as mentioned in the previous paragraph, the testing datasets, which are anomalous data, are fed into the systems and then the output, which are the corrected data, are compared with the original normal data. If they are the same, the correction coverage score is increased by 0.001. In other words, if the systems successfully correct all the anomalous data, the coverage score would be 1 since they are 1000 datasets.

5.1. Correction coverage analysis

The methods are evaluated with different window sizes and various numbers of injected anomalies. Figures 5, 6, and 7 depict the effect of number of anomalies in the correction coverage for all three proposed methods and these figures are for window sizes 3, 4, and 5, respectively; as might be expected, more anomalies make the correction harder for the system, and thus decreasing the correction coverage score. Typically, since the larger window size has more information, it helps to provide better correction coverage. In addition, the use of random functions in the case of multiple substitutions makes the correction coverage

```
Input: Training data, W, Membership Threshold Value
Output: Clusters with similar members and one representative for each

1. ClusterSet = EMPTY
2. SeqSet = Break down Training Data into overlapping sequences with length W
3. FOR ALL Sequence IN SeqSet
4.     FOR ALL Cluster IN ClusterSet
5.         IF similarity of Sequence with all members of Cluster>Membership Threshold Value THEN
6.             Add Sequence to Cluster
7.             Exit FOR
8.         END IF
9.     END FOR
10.    IF Sequence is not added to any cluster THEN
11.        Cluster = Create a new cluster
12.        Add Sequence to new Cluster
13.        Add Cluster to ClusterSet
14.    END IF
15. END FOR

16. FOR ALL Cluster IN ClusterSet
17.     Select one member of Cluster as representative
18. END FOR
19. Anomaly Threshold = Find a threshold based on given Membership Threshold Value

20. RETURN ClusterSet
```

Figure 3. The pseudo code of training phase of the proposed cluster-based anomaly correction algorithm.

```
Input: Testing Data, ClusterSet, Anomaly Threshold Value, W
Output: Corrected Testing Data

1. SeqSet = Testing data breaks down to overlapping sequences with the length W
2. FOR ALL TestingSequence IN Testing Data
3.     FOR ALL Cluster IN ClusterSet

         /*Detection*/
4.       IF Similarity(sequence, representative of Cluster) > Anomaly Threshold Value THEN
5.           IF for all members of Cluster, the similarity of Sequence > Anomaly Threshold Value THEN
6.               Mark the Sequence as normal
7.           ELSE
8.               Mark the Sequence as anomaly
9.           END IF
10.      ELSE
11.          Mark the Sequence as anomaly
12.      END IF
13.      IF Sequence is anomaly THEN

             /* Correction*/
14.          FOR ALL Cluster IN ClusterSet
15.              FOR ALL Member IN Cluster
16.                  Find a Member with maximum similarity to Sequence
17.                  Correct Testing Data using substitution of Sequence with Member
18.              END FOR
19.          END FOR
20.      END IF
21.  END FOR
22.END FOR

23. RETURN Testing Data
```

Figure 4. The pseudo code of testing phase of the proposed cluster-based anomaly correction algorithm.

be below perfect. In other words, since the system randomly selects a substitution based on the weights of the choices, the results will not be 100 percent, overall. Moreover, Figures 5, 6, and 7 show the performance of the proposed methods against each other. It shows that the Markov-based correction method performs better in the matters of anomaly correction coverage for all different window sizes.

In addition, various similarity functions are implemented to obtain the best similarity function for

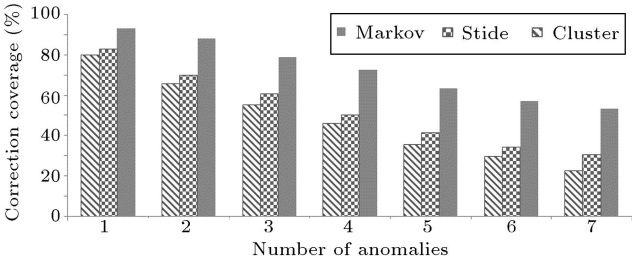

Figure 5. Correction coverage of all three proposed methods (window size = 3).

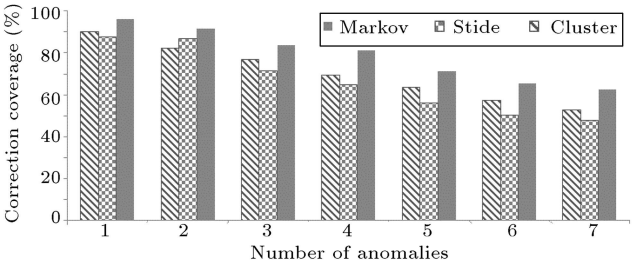

Figure 6. Correction coverage of all three proposed methods (window size = 4).

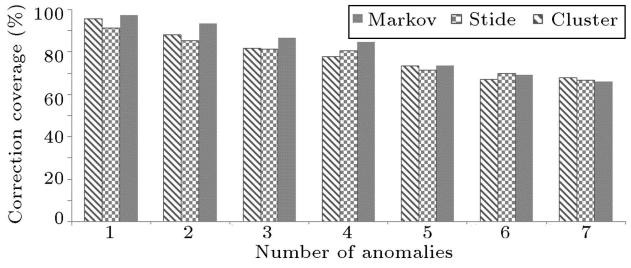

Figure 7. Correction coverage of all three proposed methods (window size = 5).

anomaly correction. The results of the correction coverage of Cluster-based anomaly correction with different similarity functions are shown in Tables 8, 9, and 10. These tables present correction coverage of the proposed method with window sizes 3, 4, and 5, respectively.

The results of Tables 8, 9, and 10 show that *Goodall2* is the best similarity function to correct the anomalies. Another evaluation is done to find the best *membership threshold value* to obtain a better correction coverage. Table 11 presents different correction coverages of Cluster-based anomaly detection, and correction method uses similarity function *Goodall2* with various *membership threshold values*. The best *membership threshold value* is 0.5 according to the results of Table 11.

5.2. Power, area, and time consumption analyses

The proposed methods are implemented by and synthesized in design compiler. The corresponding results of power, area, and time consumption are shown in Table 12. The results are also normalized to minimum value obtained. It shows that the overheads of Markov-based method are noticeably higher than those of the other two methods are. This is due to its more information needed for constructing transition between events in comparison to Stide-based and Cluster-based methods, resulting in a better correction coverage and, as expected, more power, area and time are consumed for checking more constraints.

Similar to most digital systems, there is a trade-off between performances and overheads, in which performance is correction coverage and overheads are power, area and time consumption. The parameter of tuning this trade-off is the window size, that is, larger window size yields better correction coverage and more overheads.

All of the proposed methods are performed acceptably in small window sizes, depicting the methods that are well designed. Their general drawback is that, regardless of the overheads of the correction circuit, it

Table 8. Anomaly correction coverage (in percentage) of cluster-based anomaly detection and correction with window size 3.

Function	Correction coverage (%) for different number of anomalies						
	1	2	3	4	5	6	7
Overlap	75.48	52.87	39.72	30.45	22.05	16.9	12.58
Skin	78.74	60.01	45.32	35.98	27.74	21.28	15.74
Goodall1	75.52	51.32	37.12	29.68	21.08	14.72	8.71
Goodall2	**78.3**	**63.64**	**52.51**	**41.38**	**34.22**	**27.1**	**21.02**
Goodall3	76.18	51.00	36.04	29.27	21.51	14.2	8.75
Goodall4	20.64	16.02	13.01	8.11	6.28	5.45	3.94
Gambaryan	76.42	62.21	48.91	37.87	30.11	23.8	19.02
Burnaby	81.55	56.17	45.42	32.82	28.3	20.84	13.72

Table 9. Anomaly correction coverage (in percentage) of cluster-based anomaly detection and correction with window size 4.

Function	Correction coverage (%) for different number of anomalies						
	1	**2**	**3**	**4**	**5**	**6**	**7**
Overlap	87.12	77.02	67.92	58.57	54.32	46.98	41.31
Skin	88.75	78.58	70.51	62.52	55.32	48.05	44.81
Goodall1	88.01	75.51	62.58	58.62	53.98	46.32	40.12
Goodall2	**90.01**	**82.60**	**75.72**	**67.50**	**61.97**	**57.45**	**51.38**
Goodall3	88.64	75.92	67.60	58.92	54.17	46.73	41.47
Goodall4	30.21	24.81	22.91	17.97	15.80	14.61	12.24
Gambaryan	87.54	78.57	70.12	61.84	54.30	49.07	43.27
Burnaby	92.17	81.81	75.41	65.61	61.42	57.08	50.71

Table 10. Anomaly correction coverage (in percentage) of cluster-based anomaly detection and correction with window size 5.

Function	Correction coverage (%) for different number of anomalies						
	1	**2**	**3**	**4**	**5**	**6**	**7**
Overlap	91.01	81.81	74.41	64.14	69.91	59.43	54.67
Skin	91.63	84.29	77.57	73.14	66.46	61.70	55.50
Goodall1	92.13	82.11	77.21	71.41	65.21	60.73	53.59
Goodall2	**96.14**	**88.39**	**82.13**	**78.34**	**72.61**	**68.99**	**65.80**
Goodall3	91.89	82.59	76.26	72.11	65.69	61.24	52.64
Goodall4	33.73	28.89	27.63	23.24	20.91	21.11	19.43
Gambaryan	94.34	86.49	79.37	74.79	68.71	63.67	60.71
Burnaby	97.37	90.14	85.21	81.34	76.63	72.44	66.81

Table 11. Anomaly correction coverage (in percentage) of cluster-based anomaly detection and correction with different window sizes and membership threshold values.

Membership threshold	Correction coverage (%) for different number of window size			
	3	**4**	**5**	**Average correction coverage (%)**
0.0	46.70	69.86	78.40	64.99
0.2	46.56	70.14	79.13	65.28
0.4	46.94	70.24	78.60	65.26
0.5	**47.74**	**70.36**	**78.89**	**65.66**
0.6	41.67	70.67	79.26	63.87
0.8	41.60	65.11	79.91	62.21
1.0	46.99	70.27	78.21	65.16

forces some delay in delivering the data to the controller since the system needs to analyze all events in the range of the window. All in all, based on the application of the correction coverage, it can be tuned based on the priorities using the window size.

5.3. Cost-effective correction coverage

In order to find the best anomaly correction method, which can detect and correct most of anomalies in testing data and also impose the least overhead, we proposed a metric which is called Cost-Effective Cor-

Table 12. Hardware consumption of anomaly correction methods for different window sizes.

Window size	Methods	Area		Dynamic power		Leakage power		Delay	
		(μm^2)	Normal	(μw)	Normal	(μw)	Normal	(ns)	Normal
3	Markov-based	249.64	120.20	62.85	3142.50	1.87	6.23	3.98	66.33
	Side-based	63.35	30.46	3.96	198.00	0.4	1.33	0.87	14.50
	Cluster-based	**2.08**	**1**	**0.02**	**1**	**0.3**	**1**	**0.06**	**1**
4	Markov-based	415.48	153.88	87.43	4371.50	1.91	6.16	4.12	51.50
	Side-based	239.61	88.74	39.72	1986.00	0.11	0.35	1.39	17.38
	Cluster-based	**2.7**	**1**	**0.02**	**1**	**0.31**	**1**	**0.08**	**1**
5	Markov-based	10787.73	2580.80	814.89	5820.64	96.21	2405.25	18.8	56.97
	Side-based	348.58	83.39	59.91	427.93	0.18	4.50	1.6	4.85
	Cluster-based	**4.18**	**1**	**0.14**	**1**	**0.04**	**1**	**0.33**	**1**

rection Coverage (CECC). This metric is defined as follows:

$$(\text{CECC}) = \frac{\text{Correction Coverage}}{\text{Area} \times \text{Power} \times \text{Delay}}. \quad (5)$$

In safety-critical embedded systems which require more correction coverage and also need to have the least area, power and also delay overheads, this metric should be used as much as possible. Therefore, the methods are re-evaluated based on CECC metric and the results are mentioned in Table 13. Based on the results in this table, Cluster-based method shows the best performance in terms of having the best fault-tolerant property (correction coverage) versus other overheads. This is due to the trading-off between advantages and disadvantages of the method, showing better advantages compared to its disadvantage.

6. Conclusions

This paper provides three methods to correct the anomalies in embedded systems in order to make them more fault-tolerant. These methods show good performance with small window size in categorical data. Table 14 shows a general comparison between the methods discussed in this paper, where symbols "+" and "-" depict holding the attributes. The main implications of this paper are as follows:

- It is important in embedded systems to not only identify the faults, but also to recover from them;
- It is possible to construct an anomaly correction method based on anomaly detection one;
- Markov modeling can be also used in anomaly correction;

Table 13. Cost effective correction Coverage for different anomaly correction methods versus different window sizes.

Window size	Methods	Average correction coverage		Cost Effective Correction Coverage (CECC)	
		(%)	Normal	($1/\mu m^2/\mu w$/ns)	Normal
3	Markov-based	72.5	1.57	1.12E-3	9.79E-7
	Side-based	50.2	1.09	2.08E-1	1.81E-4
	Cluster-based	**46**	**1**	**1151.82**	**1**
4	Markov-based	81.1	1.17	5.30E-4	5.45E-7
	Side-based	65	0.93	4.90E-3	5.03E-6
	Cluster-based	**69.4**	**1**	**973.63**	**1**
5	Markov-based	84.5	1.09	4.57E-7	1.46E-9
	Side-based	80.5	1.03	2.40E-3	7.68E-6
	Cluster-based	**77.7**	**1**	**312.94**	**1**

Table 14. Analyzing of mentioned different methods.

Methods		Detection	Correction	Area overhead	Correction vs. overhead trade-off
Markov [9]		+	−	++	−
Stide [9]		+	−	+	−
Probability-based [14]		+	−	+	−
Buffer-based [14]		+	−	+	−
Introduced correction methods	Markov-based	+	++	++	+
	Stide-based	+	+	+	+
	Cluster-based	+	++	+	++

- In the Cluster-based anomaly correction, it was shown that the similarity of the events has enough information to be considered for anomaly correction;
- In the Stide-based anomaly correction, it is still able to perform adequately, even though it has less information about the data in comparison with Markov-based method.

References

1. Lovato, A.V., Araujo, E. and da Silva, J.D.S. "Fuzzy decision in airplane speed control", *IEEE International Conference on Fuzzy Systems*, pp. 1578-1583 (2006).
2. Patcha, A. and Park, J.M. "An overview of anomaly detection techniques: Existing solutions and latest technological trends", *Computer Networks*, **51**(12), pp. 3448-3470 (2007).
3. Spence, C., Parra, L. and Sajda, P. "Detection, synthesis and compression in mammographic image analysis with a hierarchical image probability model", In *Mathematical Methods in Biomedical Image Analysis*, 2001. MMBIA 2001. IEEE Workshop on, 2001, pp. 3-10 (2001).
4. Imran, N., Jooheung, L., Youngju, K., Mingjie, L. and Ronald, F.D. "Amorphous slack methodology for autonomous fault-handling in reconfigurable devices", *International Journal of Multimedia & Ubiquitous Engineering*, **7**(4), pp. 29-44 (2012).
5. Dilillo, L., Alberto, B.V., Miroslav, G., Patrick, P. Serge, and Arnaud, V. "Error resilient infrastructure for data transfer in a distributed neutron detector", In *IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*, pp. 294-301 (2011).
6. Kaur, K. and Singh, E.N. "A survey of intrusion detection techniques", *International Journal of Advanced Research*, **3**(6), pp. 402-405 (2013).
7. Bicego, M., Murino, V., Pelillo, M. and Torsello, A. "Similarity-based pattern recognition", *Pattern Recognition*, **39**(10), pp. 1813-1814 (2006).
8. Maxion, R.A. and Tan, K.M.C. "Anomaly detection in embedded systems", *IEEE Transactions on Computers*, **51**(2), pp. 108-120 (2002).
9. Aleskerov, E. and Rao, B. "A neural network based database mining system for credit card fraud detection", In *Computational Intelligence for Financial Engineering (CIFEr)*, pp. 220-226 (1997).
10. Chandola, V., Banerjee, A. and Kumar, V. "Anomaly detection: A survey", *ACM Computing Surveys (CSUR)*, **41**(3), p. 15 (2009).
11. Quanz, B., Fei, H., Huan, J., Evans, J., Frost, V., Minden, G., Deavours, D., Searl, L., DePardo, D., Kuehnhausen, M., Fokum, D., Zeets, M. and Oguna, A. "Anomaly detection with sensor data for distributed security", *Proceedings of 18th International Conference on Computer Communications and Networks*, pp. 1-6 (Aug. 2009).
12. Mojarad R. and Zarandi H.R. "Two effective anomaly correction methods in embedded systems", *CSI Symposium on Real-Time and Embedded Systems and Technologies (RTEST)*, pp. 68-76 (2015).
13. Mojarad, R. and Zarandi, H.R. "Markov-based anomaly correction in embedded systems", *International Journal of Computer Theory and Engineering*, **8**(4), pp. 272-2979 (Aug. 2016).
14. Mojarad, R., Kordestani, H. and Zarandi, H.R. "A cluster-based method to detect and correct anomalies in sensor data of embedded systems", *24th Euromicro International Conference on Parallel, Distributed, and Network-based Processing (PDP)*, pp. 240-247 (2016).
15. Branch, J.W., Giannella, C., Szymanski, B., Wolff, R. and Kargupta, H. "In-network outlier detection in wireless sensor networks", *Knowledge and Information Systems*, **34**(1), pp. 23-54 (2013).
16. Hill, D.J. and Minsker, B.S. "Real-time Bayesian anomaly detection for environmental sensor data", In *Proceedings of the Congress-International Association for Hydraulic Research*, **32**(2), p. 503 (2007).
17. Du, W., Fang, L. and Peng, N. "Lad: Localization anomaly detection for wireless sensor networks", *Journal of Parallel and Distributed Computing*, **66**(7), pp. 874-886 (2006).

18. Amir, A., Zimet, L., Sangiovanni-Vincentelli, A. and Kao, S. "An embedded system for an eye-detection sensor", *Computer Vision and Image Understanding*, **98**(1), pp. 104-123 (2005).

19. Zandrahimi, M., Zarandi, H.R. and Mottaghi, M.H. "Two effective methods to detect anomalies in embedded systems", *Microelectronics Journal*, **43**(1), pp. 77-87 (2012).

20. Hamilton, J.D., *Time Series Analysis*, Princeton: Princeton University Press (1994).

21. Hofmeyr, S.A. Forrest, S. and Somayaji, A. "Intrusion detection using sequences of system calls", *Journal of Computer Security*, **6**(3), pp. 151-180 (1998).

22. Das, K. and Schneider, J. "Detecting anomalous records in categorical datasets", In *Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 220-229 (2007).

23. http://www.cs.unm.edu/immsec/data/synthsm.html (Sep. 2011).

24. Bhatnagar, H. "Advanced ASIC chip synthesis using synopsys design compiler", *Physical Compiler and PrimeTime*, Springer (2002).

25. http://www.tkt.cs.tut.fi/tools/public/tutorials/ synopsys/design_compiler/gsdc. html, Date Visited: Aug. 2013.

Biographies

Roghayeh Mojarad received her BS degree in Computer Hardware Engineering from University of Tehran in 2011. She continued studying the same field and managed to receive her MS degree in Computer Engineering from Department of Computer Engineering and Information Technology, Amirkabir University of Technology (Tehran Polytechnic) in 2013. She was recognized as the top student in her graduate school. She is currently a PhD student in University Paris-Est Creteil (UPEC). Her research interests include answer set programming, activity recognition, fault-tolerant systems, digital system design, and FPGA architecture design.

Hamid Reza Zarandi received his PhD, MS, and BS degrees in Computer Engineering all from Sharif University of Technology, Iran, in 2006, 2002, and 2000, respectively. He is currently an Associate Professor of Computer Engineering and Information Technology Department at Amirkabir University of Technology (Tehran Polytechnic). His research interests include high-performance computing, dependable computer architecture, fault-tolerant computing and embedded systems on which he published more than 100 refereed conference and journal papers. Dr. Zarandi is a member of IEEE Computer Society and Computer Society of Iran.