Detecting Anomaly in the Usage of Database Attribute

Kaiping Liu, Hee Beng Kuan Tan, Arnatovich Yauhen Leanidavich
School of Electrical and Electronic Engineering, Nanyang Technological University
Block S2, Nanyang Avenue, Singapore
{kpliu, ibktan}@ntu.edu.sg, YAUHEN001@e.ntu.edu.sg

Abstract— In database applications, database operations should be provided to maintain persistency of the data which is often represented as database attributes. Any missing, redundant or inconsistent operation performed on database attributes would indicate anomaly or even program bugs. Through characterizing operations performed in database transactions on database attributes, we extract a feature vector from code for each attribute. This paper proposes a clustering-based approach which analyzes the feature vectors to automatically detect anomalies in the usage of database attributes. Once an anomaly is detected, developers can perform investigation to take corrective actions if necessary. The evaluations on both industrial and open source database applications show that our approach is able to detect many types of anomalies in the usage of database attributes with a high detection rate (92.8% on average), and a low false positive rate (0.57% on average).

Keywords—anomaly detection; database application; clustering; attribute usage.

I. INTRODUCTION

Database is a major component of many software systems. In database applications, adequate operations should be provided to maintain the persistency of data. Any missing, redundant or inconsistent operation performed on database attributes would indicate anomaly of a database application.

Anomalies are instances that cannot be classified under any normal behavior. The PHP code snippet shown in Fig. 1, is an example of anomaly in database operations.

In Fig.1 line 6, the attribute “family_name” is selected in order to update the attribute “family_name” in table `tb_user_info` in line 17. However, from the database schema, we found that the table “tb_user_info” requires non-null value for the attribute “family_name” but the table “tb_user_account” does not have such requirement. Furthermore, from investigation of the code, there is no program define the value of the attribute “family_name”. So the result of the SELECT operation contains a null value. Hence, the execution of the UPDATE operation would cause exception. It is desirable to detect such kind of anomaly. Our work aims to address this issue.

This paper proposes an approach to detect anomalies in the use of database attributes by means of abstracting and characterizing database operations. We present a clustering-based anomaly detection algorithm, which takes as inputs a set of unlabeled database attributes and finds anomalies within them. We make two general assumptions: 1) the number of normal attributes in the training set greatly exceeds the number of anomalous attributes, thus implying that the normal attributes should form larger clusters compared to the anomalous attributes; 2) attributes having the same type of anomaly should be close to each other in feature space under some reasonable metric, while attributes with different type of anomaly will be far apart. The above assumptions mean that any anomalous attribute would appear to be an outlier in the dataset due to its rarity and abnormality.

Figure1. PHP code example

Our approach groups the database attributes together into clusters using a distance-based metric. Once the database attributes are clustered, we identify small clusters and label them as anomalous clusters. Cross-project validations are performed on three industrial database applications and seven open source database applications to verify the approach. The results show that our approach is able to detect many types of anomalies with an average detection rate 93.0%, while maintaining a low false positive rate.

The paper is organized as follows. Section II introduces the extraction of the feature vector for attribute usage. Section III describes our approach. Section IV evaluates the proposed approach and reports the experiment result. Section V
discusses the performance of our approach. Section VI discusses the related work. Section VII concludes the paper.

II. CHARACTERIZING DATABASE OPERATIONS USING FEATURE VECTORS

A. Database Operations

Since transaction is the atomic processing unit in a database application, we characterize the database operations performed on database attributes on a per-transaction basis. We classify the database operations into the following types:

- **Create (C):** A value of an attribute is inserted.
- **Null Create (NC):** A record that contains the attribute is inserted without defining the value of the attribute.
- **Control Update (COU):** The value of an attribute is updated by a new value that is not influenced by the existing attribute value and inputs from user and database.
- **Overriding Update (OVU):** The value of an attribute is updated to a new value that is influenced by user input but is not influenced by the existing value of the attribute.
- **Cumulating Update (CMU):** The value of an attribute is updated to a new value that is influenced by the existing attribute value.
- **Delete (D):** The value of an attribute is deleted as a result of deletion of a record containing it.
- **Use (U):** The value of an attribute is used to support the insertion, updating or deletion of other database attributes or output to the external environment. It is not considered under this case if the value of an attribute is used to update itself.
- **Other Update (OU):** If there are multiple database operations performed on a database attribute in one transaction, the database operation is classified as Other Update.

We characterize the usage of a database attribute using an eight-element Boolean vector \([m_1, m_2, m_3, m_4, m_5, m_6, m_7, m_8]\), where \(m_1, m_2, m_3, m_4, m_5, m_6, m_7, m_8\) denote the existence of a transaction in which an operation performed on the attribute is of type C, NC, COU, OVU, CMU, D, U or OU respectively. This vector is called feature vector for attribute usage.

B. Extraction of Feature Vector for Attribute Usage

To automatically extract the database operations performed on each attribute in a transaction, static program analysis is performed on each path in a database application. Since PHP is a widely used scripting language, this paper uses PHP programs as subjects for demonstrations and experiments. However, the approach can be applied to any database application as it is based on basic control and data dependency analysis with regard to standard SQL statements.

1) Query extraction

In a database application, the SQL queries are commonly formed by dynamically concatenating several string literals and variables. From the inter-procedural control flow graph of the example code in Fig. 1, two paths can be found that perform database transactions, and both of which start from the MAIN function’s entry point, and reach the SQL execution function “do_query”. The actual queries executed in these two transactions are different.

We follow the technique used in structural program testing by generating a set of basis paths using the baseline method proposed in [1]. Specifically, we traverse the loop body only once. For each path in the basis set, whenever we encounter a query execution function like “mysql_query”, the definition of every part of its parameter which is a query string is retrieved and concatenated in order. As an example, for the code in Fig. 1, the extracted queries of the SELECT operation would be: 1) "SELECT family_name as fn FROM tb_user_account WHERE user_id='__';" 2) "SELECT family_name as fn FROM tb_user_account WHERE user_id=0".

2) Extraction of Feature Vector

After the queries are extracted, we analyse each query to obtain the characteristics of database operations using an SQL grammar parser. All the CREATE TABLE queries are first parsed. Then, we analyse the queries according to their operation types as follows:

- **SELECT:** The SELECT query is parsed, table aliases are identified and restored by the actual table names, and the attributes are identified. The attribute names are extracted from the select list, JOIN expressions as well as the WHERE clause. The star-shorthand “*” is regarded as referencing all attributes.
- **INSERT:** After parsing, the table name is identified. We then check whether there is column list in this query. When no column list is provided, it is assumed that values of all the attributes in this table are to be inserted. Those attributes declared in the schema as “auto incremental” or having not-null default values are also viewed as inserted by the query.
- **UPDATE:** We not only collect the attribute names that are updated in one query, but also identify the update pattern for each attribute. We analyze the value string to determine the update type, i.e. either COU, OVU or CMU for the attribute. For example, the assignment expression name=$_POST['username'] is an example of OVU, whereas the expression balance=balance-$withdraw is an example of CMU.
- **DELETE:** We identify the table name, and mark all the attributes of this table as “Delete”.

Besides, the attributes in the WHERE clause are characterized as “Use”. If an attribute has multiple operations in this transaction it will be marked as OU in this transaction; otherwise, it will be marked as one of the other seven features as described in Section II.A. After all transactions have been analyzed, the characteristics of database operations performed on each attribute are merged to generate its feature vector. For example, if an attribute experiences the type “Create” for at least one time, the first element of the feature vector would be 1; otherwise, it is 0.
III. DETECTING ANOMALIES IN THE USAGE OF DATABASE ATTRIBUTES

The extracted feature vectors are filtered and passed to the clustering algorithm. We use the training data to estimate the optimal parameters values. Based on the clustering result and optimal parameters values, normal and anomalous clusters are identified. Then the model can be used to detect anomalies in a new database application. The overview of the approach can be seen in Fig. 2.

A. Clustering Method

The extracted feature vectors have a relatively low dimensionality and the elements of the vector are binary value. Hence, we adopt a simple variant of single-linkage clustering method which proceeds as follows: given a fixed metric \( M \) and a constant cluster width \( W \), the distance between \( C \)'s defining instance \( C \) and \( d \) can be computed as Euclidean \( \text{dist}(C,d) \). Here \( C \) is the feature vector that defines the centroid of that cluster. If the distance is within \( W \), the instance \( d \) will be assigned to the nearest cluster. Otherwise, a new cluster would be generated and the attribute feature vector would become its centroid. The Euclidean distance used to measure the distance between a cluster \( C \) and an instance \( d \) is defined as:

\[
\text{dist}(C,d) = \sqrt{\sum_{i=1}^{n} (c_i - d_i)^2}
\]

where \( n \) is the dimension of the vectors, and \( c_i \) is the element of the centroid of \( C \) while \( d_i \) is the element of vector \( d \).

B. Labeling Clusters

After clustering, it remains unknown which clusters consist of normal attributes and which ones contain anomalies because the attributes are unlabelled during clustering. In our work, we assume that an overwhelming majority of the training dataset is made up of normal attributes and small clusters are likely to consist of anomalies. Hence, we label the top \( P\% \) largest clusters as ‘normal’, and the remaining clusters as ‘anomalous’.

C. Detection of Anomalies

Once the clusters are created and labelled, they can be used to detect anomalies in the use of database attributes. Given a new database application, we can extract the feature vector for each attribute. For each feature vector \( v \), we find the cluster that includes \( v \) and assign this attribute with that cluster’s label. If \( v \) does not belong to any clusters discovered thus far, a new cluster would be generated and \( v \) would become its centroid. We will manually examine \( v \) and decide it is anomalous or normal.

IV. EVALUATION AND RESULTS

A. Training Dataset Description

In our research, it was extremely difficult to find readily available dataset. Hence, we selected three matured, large-scale industrial database applications and seven open source database applications from sourceforge.net. The three selected industrial database applications come from different domains, including a management system, a school management system and a web-based e-commerce system. The open source systems include ChurchInfo, Front Accounting, CourseMS, Gliding Booking System, CiteCRM, AluminiServer and Hotel Booking Portal. We obtained a dataset with 8909 source code files, 1393K lines of code and 3302 database attributes in total. All the experiments were conducted on a desktop PC with an Intel Core Duo 2.4GHz CPU and 4GB memory.

For data collection, we have built our extraction tool on top of PHC [2], an open source PHP compiler. Our tool performs data flow analysis on the paths of the inter-procedural CFG for each system, and extracts all the database transactions from them. Feature vectors for all the attributes are then formed.

B. Performance Measurement

Two measures were computed over all labelled attributes to access the performance of our approach.

- Detection rate = the number of anomalous attributes detected by the system / the total number of anomalous attributes presented in the testing dataset;
- False positive rate = the total number of normal attributes that were wrongly classified as anomalous / the total number of normal attributes.

To calculate these values, access to labels of attributes in the dataset was required. Based on the database schema, we extracted the attributes in the database. After analysing the program code and based on the attributes extracted, we can identify whether proper and adequate operations have been conducted on each attribute and then labelled each attribute with “normal” or “anomalous”. We calculated these two measures over all labelled attributes.

C. Filtering the Training Dataset

We make use of 30% of the dataset (990 attributes) as our training dataset. Our first assumption states that the number of normal attributes greatly exceeds that of the anomalous attributes. Hence, in the resulting training dataset, the number of normal attributes (96.36%) greatly exceeds that of the anomalous attributes (3.64%).

D. Fixing Parameters

It is necessary to fix and optimize the values of two parameters first: the cluster width \( W \) and the percentage \( P \).
determines the minimum distance between two attribute feature vectors assigned to the same cluster, while the percentage \( P \) decides the top \( P\% \) largest clusters that are labelled as 'normal'.

A series of tests was conducted on the training dataset for using a range of values of the two variables \( W \) and \( P \). Table I shows the results for optimizing the values of \( W \), and their corresponding measured performances. As we use the Euclidean distance measure, the cluster width \( W \) is set to a value slightly below each Euclidean distance value (such as \( \sqrt{1}, \sqrt{2}, \ldots, \sqrt{8} \)).

<table>
<thead>
<tr>
<th>( W )</th>
<th>( P )</th>
<th>DR</th>
<th>FPR</th>
<th>( W )</th>
<th>DR</th>
<th>FPR</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.7</td>
<td>60%</td>
<td>0.419%</td>
<td>2.2</td>
<td>60%</td>
<td>58.3%</td>
<td>1.992%</td>
</tr>
<tr>
<td>1.4</td>
<td>60%</td>
<td>0.108%</td>
<td>2.4</td>
<td>60%</td>
<td>47.2%</td>
<td>4.193%</td>
</tr>
<tr>
<td>1.7</td>
<td>60%</td>
<td>0.1468%</td>
<td>2.6</td>
<td>60%</td>
<td>38.9%</td>
<td>5.241%</td>
</tr>
<tr>
<td>1.9</td>
<td>60%</td>
<td>0.1572%</td>
<td>2.8</td>
<td>60%</td>
<td>33.3%</td>
<td>6.289%</td>
</tr>
</tbody>
</table>

We decided to use \( W = 0.7 \) in subsequent tests, since it produced a low false positive rate and high detection rate. To find the value for \( P \), we conducted several tests on the same dataset. The results of some of the tests are shown in Table II.

<table>
<thead>
<tr>
<th>( W )</th>
<th>( P )</th>
<th>DR</th>
<th>FPR</th>
<th>( W )</th>
<th>DR</th>
<th>FPR</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.7</td>
<td>80%</td>
<td>72.2%</td>
<td>0.314%</td>
<td>0.7</td>
<td>40%</td>
<td>97.2%</td>
</tr>
<tr>
<td>0.7</td>
<td>70%</td>
<td>80.6%</td>
<td>0.419%</td>
<td>0.7</td>
<td>30%</td>
<td>97.2%</td>
</tr>
<tr>
<td>0.7</td>
<td>60%</td>
<td>94.4%</td>
<td>0.524%</td>
<td>0.7</td>
<td>20%</td>
<td>100%</td>
</tr>
<tr>
<td>0.7</td>
<td>50%</td>
<td>94.4%</td>
<td>6.499%</td>
<td>0.7</td>
<td>10%</td>
<td>100%</td>
</tr>
</tbody>
</table>

From table II, it can be seen that the detection rates are the same when \( P = 50\% \) or \( P = 60\% \), though the false positive rate is fairly lower when \( P = 60\% \). Therefore, \( P = 60\% \) was chosen.

The clustering results of the training dataset when \( P=60\% \), \( W=0.7 \) are shown in Table III. There are 32 clusters in total. Based on the value of \( P \), we labelled the first 19 clusters as normal and labelled the rest as anomalous. After further investigate of the anomalous attributes, we classified them into the following four types:

- **Missing database operations (MI):** For a database attribute, essential database operations (e.g., inserting a value of an attribute) are missing.
- **Inconsistent database operations (IC):** It is essential to provide an transaction in a database application to correct the effect of a transaction that has been executed with erroneous input. For a transaction that updates an attribute through cumulative update, the correction should also be made through cumulative update for control purpose. Correcting the result of a transaction that updates an attribute by “cumulative update” using “overriding update” is a common inconsistency fault.
- **Redundant database operations (RD):** For a database attribute, additional different types of database operations are performed on it.
- **No Update (NU):** For a database attribute, the program does not provide any operation to maintain or use it.

It can be seen from table IV that there are a total of 956 attributes (96.57\%) labeled as “normal” (NM) and 34 attributes (3.43\%) labeled as “anomalous” (MI, IC, RD or NU).

### Cross Validation Testing

Based on the parameters and clusters learned from the training process, we performed cross-project evaluation by using a variant of the cross validation method.

We formed five subsets from the entire dataset, each containing approximately 660 attributes. It should be noted that no information about the attributes’ label was used during the clustering in the training process, therefore, the training set still can be used for testing. Since we require that the number of anomalous attributes should constitute a very small portion of the training dataset, it was found that there were two subsets failed to meet this requirement. Therefore for cross validation training only three of the five subsets are selected. The numbers of normal and anomalous attributes of each of the three subsets are shown in Table IV.

Each time, one of the three subsets was chosen as the training dataset to perform the clustering process. Then the clusters are labeled and used to test each of the other two subsets. In this manner, the testing is conducted 6 times in total. The results are shown in Table V.

Our evaluations show that the proposed approach can achieve a high detection rate with a low false positive. We believe that this technique is capable of detecting the anomalies in attribute usage in database applications.

### Some Examples

The attribute “subject” in the table “as newsletter” in AlumniServer is predicted as missing function. Which means this attribute is neither defined nor updated. However, it is referenced and is declared as “default null” in the database schema. This could be a potential fault because null value frequently causes unexpected result or failure of the system.

In the prediction results of ChurchInfo, we found several no update attributes in table “user_usr”. This table is used to store the information of users. There are eight attributes named “usr_CalNoSchool1”, “usr_CalNoSchool12”, …, “usr_CalNoSchool8” in this table. However, only the attribute “usr_CalNoSchool1” among the eight has UPDATE operation. It is reasonable to argue that this could be inappropriate. After carefully searching the system, we spotted the following code snippet in the source code file Default.php, which updates the table “user_usr”:

```php
snippet in the source code file Default.php, which updates the table “user_usr”:
```
After investigation, we found that the developer in fact intended to update all these eight attributes according to some conditions. However, all the strings in the concatenation were likely mistyped as "usr_CalNoSchool1", resulting in no update operation for the other seven attributes. Aided by the prediction results, developers or maintainers can take corresponding actions to deal with this.

G. Threats to Validity

It can be argued that the values of the parameters we evaluated may depend on the domain. Therefore, to vindicate our approach, we have chosen database systems of different sizes and complexities from different domains. Ideally, we would like both of our assumptions mentioned in Section I to be satisfied. In reality, of course this assumption may not be fully satisfied and this is also one of the primary reasons that our method fails to detect 100% of the anomalous. However, we still believe that the proposed approach can be applied to a variety of database applications, and the best way to prove our conclusions is to replicate and extend our experiments.

V. RELATED WORK

Data mining methods have been well utilized in software engineering especially intrusion detection. Portnoy et al. [3] proposed a clustering method to detect intrusion by regarding intrusion as anomaly against normal behaviors. Eskin, et al. [4] investigated the effectiveness of three algorithms in intrusion detection. Furthermore, Eskin et al. [5] also applied machine learning method to learn a mixture probability distribution in order to model the anomalies for intrusion detection. Oldmeadow et al. [6] carried out further research based on the clustering methods mentioned in [4] and showed improvements in accuracy.

Detecting anomalies in data has been studies in the statistics community as early as the 19th century. Chandola et al. [7] provided a comprehensive survey on the anomaly detection techniques. Our work is closely related to partitioning method which constructs k partitions of a database of n objects where each partition represents a cluster. Based on the
assumption that normal data instances belong to large and dense clusters, while anomalies either belong to small or sparse clusters, several approaches have been proposed [8-11]. All these studies show that most types of anomalies in programs do exhibit certain characteristics, and therefore by mining these characteristics, we can find the anomalies effectively. Our work also shows that anomalies in database operations can be discovered through data mining methods.

Besides the above-mentioned methods, other methods have also been proposed for anomaly detection. For example, Xie et al. [12] proposed to use hidden semi-Markov model for the purpose of network DDOS detection. Vigna et al. [13] proposed to detect web-based attacks by combining analyses on HTTP request and SQL queries. Zhang et al. [14] computed the anomaly score of a data instance as the sum of its distances from its k nearest neighbors. Their method differs from ours in that they only focus on the syntactic level of the queries but not system semantic level. Our work shows that anomalies in database operations can be discovered through data mining methods.

Over the years, a lot of work has been done to investigate how to automatically detect faults [15-18]. In their methods, they adopted both the supervised and unsupervised pattern classification and multivariate visualization. Different from the above-mentioned approaches, our approach is to mine anomalies in database attribute usage from source code using unlabelled feature vectors. Our approach is able to efficiently detect anomalies in database applications. We have applied the proposed approach to ten large-scale database applications. These applications contain around 8900 source files and more than 1390KLOC. Hence, we believe our approach can be applied to large-scale database applications. Furthermore, our approach can be used in the verification and validation of database applications. To date, we believe that no such approach has been proposed before. In opposing to our earlier work on data lifecycle [19], this paper applies data mining techniques to systematically detect anomalies in database attribute usage, instead of using heuristic rules.

VI. CONCLUSION

In this paper, we have presented an approach for detecting anomaly in the use of database attributes based on feature vectors extracted from database applications. The proposed approach is able to detect the anomalies of attribute usage for database applications with high accuracy while keeping the false positive rate reasonably low. On average, the detection rate is 92.8%, and the false positive rate is 0.57%. The proposed approach can be used in the verification and validation of database applications.

In future, we will conduct comprehensive experiments on a larger set of industry database applications to further validate the merits of the proposed approach. Currently, we are labeling the clusters based on the parameters we defined. Another possibility would be to label clusters which are outliers in the feature space as anomalous. Furthermore, we will also try to make the process of selecting optimal value of the parameter p automatically, by using statistical knowledge such as the mean value of the number of attributes in clusters.

REFERENCES