Deriving Complex Periodic Patterns from Nested Logs of Events

Janusz R. Getta and Marcin Zimniak

Abstract—**Periodic processing of software components is a simple reflection of a periodic nature of many real world processes where the identical actions are repeated at every given period of time. Discovering periodic patterns in the traces of computations performed in the past allows for better preparation of software systems to meet the demands of high workload times in the future.**

This work shows how to discover the periodic patterns in the nested logs of computations using the transformations of the simple patterns into the complex ones. The paper defines a concept of periodic pattern and its validation in a reduced nested log of events. A system of derivations rules is defined and followed by a sequence of algorithms that use the rules to create the complex periodic patterns. The paper is concluded with a description of experiment where the sequences SQL statement are transformed into the expressions of extended relational algebra and used as input data to discover the periodic patterns with a method described in the paper.

Index Terms—**Periodic patterns, derivation rules, nested events, nested logs.**

I. Introduction

A dynamic performance tuning allows for detection and elimination of workload peaks, performance bottlenecks, low process throughput, long response time, and the other performance related problems at run-time of a software system. Effectiveness of dynamic performance tuning strongly depends on our abilities to correctly predict the future behavior of a software system. If we know that certain data processing tasks are repeated every given period of time then to some extent it is possible predict the future behavior of a system. Information about the characteristics of the future workloads allows for preparation of a software system during the low workload times to more efficient processing during the high workload time. A typical example is an automated physical database design and performance tuning where information about the periods of exceptionally low and high workload is used by a database system to restructure data, to reduce access paths, to cluster the groups of related data, and all what prepares a database system for more efficient processing in the future [1].

The workload peaks and longer periods of very high or very low workload are caused by the interferences of periodically performed processes and randomly occurring ad-hoc processes reflecting the real world random events. One of the ways how workload peaks can be anticipated is through the discovery of periodic patterns in the processing of software components. Such patterns include information about the sequences of processed software components obtained from the aggregations of user application logs and event logs with the dynamic profiles of software components implementing a system.

Direct discovery of periodic patterns is a complex task due to several dimensions over which the patterns can be searched for and due to a large size of each dimension. For example, a periodic pattern can be defined in a multidimensional space of many software components, range dimension that determines the locations of periodic patterns in a workload history, and periodicity dimension that determines the repetition characteristics of a pattern. For large software systems a multidimensional space of candidate patterns is still finite, however, in practice it is too large for the efficient generation and verification of complex patterns.

A solution proposed in this work is based on an idea that it would be much easier and faster to find the simple periodic patterns first, and to ”derive” the complex periodic patterns from the elementary ones later on. The idea is justified by the simplicity of candidate periodic patterns selection process and pretty simple set of derivation rules. The solution reduces a dimension of complex structures of periodic patterns to the elementary patterns based on processing of a single software components and it also reduces periodicity dimension to the fixed and restricted in size periods between the repetitions of simple components. A stage of discovery of elementary periodic patterns is followed by the applications of derivation rules to create complex periodic patterns that span over the longer periods of time and that have complex repetition characteristics. The derivations provide both “sure” patterns that do not need to be verified in an audit trail as well as candidate patterns whose verification needs access to the selected parts of workload history. The outcomes from such derivations can also be compared with the contents of recorded logs in order to eliminate ”noise” created by the random processing of accidental applications.

In this work we adopt a multilevel model of historical information used for periodic process mining in [2]. At the topmost level we consider the sequences of processes performed by the user applications. At a lower level, we consider the sequences of events performed by the processes and recorded in the event logs. We generalize a concept of *event* to represent processes, events, and operations recorded at the different levels of traces, logs, and dynamic profiles. Next, we define a model of *nested log of events* that generalizes the applications traces, event logs, and dynamic

Manuscript received April 30, 2015; revised September 8, 2015.

J. R. Getta is with the School of Computing and Information Technology, University of Wollongong, Wollongong, NSW 2530, Australia (e-mail: jrg@uow.edu.au).

M. Zimniak is with Faculty of Computer Science, TU Chemnitz, Chemnitz, Germany (e-mail: marcin.zimniak@cs.tu-chemnitz.de).

profiles. We show, that it is possible to transform the application traces, events logs, and dynamic profiles into a nested event log through comparison of timestamps collected from application traces, event logs and dynamic profiles. Data preparation starts from partitioning of a period of time over which a nested event log was recorded into the disjoint time units. The nested events from the application traces, logs, and dynamic profiles are extracted from the logs and stored in an event table, which represents the hierarchies of events. Each event in the table is associated with a set of timestamps determining the moments in time when the respective complex or elementary events have been computed. An event table is further reduced by elimination of the events that inherit their properties from their parent events. In the next stage we use timestamps associated with the events in a reduced table of events and a sequence of disjoint time units to create a workload trace that contains information about the total number of times each event was processed in each one of the assumed time units. We propose a number of derivation rules that allow to discover elementary periodic patterns in a workload trace and to compose them into the complex patterns. A sequence of algorithms applies the derivation rules to discover the new periodic patterns.

The paper is organized in the following way. The next section reviews the research works related to processes mining, dynamic profiling and discovering periodic patterns. Section III defines a model of nested events, time units, and workload histograms. In Section IV we present how an event table can be created from a sequence of nested events and how it can be reduced through elimination of events that inherit their patterns from the parent events. Section V defines a concept of periodic pattern and its validation in a workload histogram. The derivation rules are presented in a Section VI and application of the rules to discovery of complex periodic patterns is included in a Section VII. An experimental implementation of the algorithms and their application to analysis audit trails in database systems is described in a Section VIII. Finally, Section IX concludes the paper.

II. Previous Work

Elimination of performance bottlenecks from software systems through dynamic profiling and analysis of collected data has been proposed in [3]. Many different types of profiles were tried to analyze the behavior of software systems [4] and to improve data flows [5]. A work [6] investigated aggregation of the results extracted from a number of large dynamic profiles and considered unification of different representations of dynamic profiles [7] and [8]. More technical information on dynamic profiling can be found in [9].

A starting point to many research studies on discovering periodic patterns, cycle pruning, cycle skipping, cycle elimination heuristics was a research presented in [10]. The works on mining frequent episodes [11], and on mining complex episodes [12] extended the works on periodic patterns.

Discovering periodic patterns in event logs appears to be quite similar to periodicity mining in time series [13] where the long sequences of elementary data items partitioned into a number of ranges and associated with the timestamps are analyzed to find the cyclic trends. However, due to the internal structures of complex events its analysis cannot be treated in the same way as analysis of sequences of atomic data items like numbers of characters.

The latest works on discovering periodic patterns address the concepts of full periodicity, partial periodicity, perfect and imperfect periodicity [14] and the most recently asynchronous periodicity [15] and [16]. The works [17] and [18] review a class of data mining techniques based on analysis of ordered set of operations on data performed by the user applications. The model of periodicity considered in this paper is a variant of the model introduced in [2] for periodic processes. An initial idea and the first system of derivation rules for the periodic patterns have been proposed in [19].

III. Nested Logs

A nested log of events is a trace of software system where the topmost level contains information about logical components of a software system and the lower levels record behavior of the lower level software components. In this work, we generalize information about the processing of logical and lower level software components into a class of *nested events*. A nice example of a nested log of events is a trace from processing of SQL statements by a database system. Start of processing of SQL statement forms the topmost level in a nested log and the operations on data sets and individual records up to data blocks form the lower levels in the log.

We consider a period of time $<t_{start}, t_{end}>$ over which a nested log event is recorded. The period of time is divided into a contiguous sequence of disjoint and fixed size *elementary time units* $<t_e^{(i)}, \tau_e>$ where $t_e^{(i)}$, for $i=1,\dots,n$ is a timestamp when an elementary time unit starts and τ_e is a length of the unit. The period $<t_{start}, t_{end}>$ consists of elementary time units such that $t_{start} = t_e^{(1)}$, and $t_e^{(i+1)} = t_e^{(i)} + \tau_e$ and $t_e^{(n)} + \tau_e = t_{end}$.

A time unit $<t, \tau>$ consists of one or more consecutive elementary time units. A nonempty sequence U of n disjoint time units $<t^{(i)}, \tau^{(i)}>$ $i=1,\dots, n$ over $<t_{start}, t_{end}>$ is any sequence of time units that satisfies the following properties: $t_{start} \le t^{(1)}$ and $t^{(i)} + \tau^{(i)} \le t^{(i+1)}$ and $t^{(i)} + \tau^{(n)} \le t_{end}$.

As a simple example consider a nested log that starts on $t_{01:01:2007:0:00am}$ and ends on $t_{31:01:2007:12:00pm}$. Then, a sequence of disjoint time units called as *morning tea time* consists of the following units $<t_{01:01:2007:10:30am}, 30>$, $<t_{02:01:2007:10:30am}, 30>, \dots, <t_{31:01:2007:10:30am}, 30>$.

Let e be a unique identifier of an event. A *nested log* of processing an event e recorded in a time unit t is denoted by $L(e{:}t)$ and it is a finite and possibly empty sequence of pairs $<e_1{:}t_1, L(e_1{:}t_1)>, \dots, <e_n{:}t_n, L(e_n{:}t_n)>$ where each e_i is a unique identifier of an event, t_i is a time unit when processing of an event e_i has started, and $L(e_i{:}t_i)$ is a nested log of event e_i recorded in a time unit t_i. All time units in a log $L(e{:}t)$ are pairwise disjoint. An empty log is denoted by $\emptyset$.

As a simple example consider the nested logs $L(e_1{:}t_1)$, $L(e_2{:}t_2)$, and $L(e_2{:}t_3)$ visualized in a Fig. 1 below. In the example a nested log $L(e_1{:}t_1)$=$<e_{11}{:}t_1,\emptyset>$, $<e_{12}{:}t_2,L(e_{12}{:}t_2)>$,$<e_{12}{:}t_3,L(e_{12},t_3)>$, a nested log

$L(e_2{:}t_2)$=<$e_{11}{:}t_3{:}$ Ø >, a nested log $L(e_2{:}t_3)$=<$e_{11}{:}t_4{:}$Ø >, <$e_{13}{:}t_5{:}$Ø >, a nested log $L(e_{12}{:}t_2)$=<$e_{121}{:}t_2$, Ø >, and a nested log $L(e_{12}{:}t_3)$=<$e_{121}{:}t_3$, Ø >.

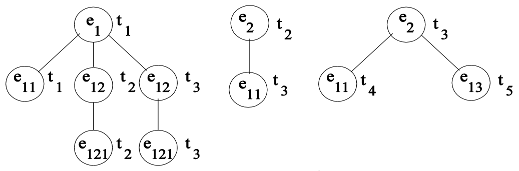

Fig. 1. The sample structures of nested event log.

Let E be a set of occurrences of events $\{e_1{:}t_1, \ldots, e_n{:}t_n\}$. A nested log of E is defined as $\cup_{e_i:t_i \in E} L(e_i{:}t_i)$ and it is denoted by $L(E)$.

A simple observation, that majority of the events usually has the same internal structure allows for compression of large nested logs. For example, due to processing plan stability required by database administrators processing of the same SQL statement in different applications accessing the same relational tables is usually performed in the same way. At the beginning of data preparation stage we transform a *nested log* into more compact *event table* and later on we reduce the table to eliminate events that have the same properties.

A definition of an event table is based on a concept of *multiset*.

A *multiset M* is a pair <S, f > where S is a set of values and f: $S \rightarrow N^+$ is a function that determines multiplicity of each element in S and N^+ is a set of positive integers [20]. In the rest of this paper we shall denote a *multiset* <e_1,..., e_m , f > where $f(e_i) = k_i$ for i=1,...,m as $(e_1^{k_1}, \ldots, e_m^{k_m})$. We shall denote an empty multiset <Ø , f > as Ø . We shall abbreviate a single element multiset (e^k) as e^k.

A *signature of an event e* is defined as a multiset of events included in a nested log $L(e{:}t_e)$ and it is denoted by $\sigma(L(e{:}t_e))$.

For example, $\sigma(L(e_1{:}t_1)) = (e_{11}, e^2_{12})$ in a nested log visualized in a Fig. 1 above.

It is important to note, that the same event recorded in the same or different logs may have different signature due to the different computational constraints imposed in the different moments in time. For example, the computations of polymorphic methods in object-oriented programming languages may return different signatures, or the computations of `SELECT` statement may return different signatures depending on the parameters of a query optimizer and the present state of a database. In the example above, the signatures of an event e$_2$ are different, i.e. $\sigma(L(e_2{:}t_2)) = (e_{11})$ and $\sigma(L(e_2{:}t_3)) = (e_{11}, e_{13})$.

If an event e_i contains in its signature an event e_j then e_i is called as a *parent event* of an event e_j. For example, the events e$_1$ and e$_2$ are the parent events of event e_{11}.

An *event table* eliminates from a nested log all repetitions of events which have identical signatures and it is defined as a set of triples <e, T, $\sigma(e)$> where e is an event, T is a set of time units when the event had occurred, and $\sigma(e)$ is a set of signatures of the event. An event table for a nested log visualized in Fig.1 is included in Table I.

The following property allows for further compression of event table. Consider an event e_i which has only one signature $(e_{i_1}^{n_1}, \ldots, e_{i_k}^{n_k}, \ldots, e_{i_m}^{n_m})$. If an event e_{i_k} has only one parent event e_i then it means an event e_{i_k} happens only when an event e_i happens. Therefore, an event e_{i_k} has the same structure of repetitions in a nested log as an event e_i and because of that it can be eliminated from an event table. For example, in an event table above, a multiset of parent events of an event e_{12} is equal to (e_1) and only one event e_1 has been recorded in a log. It means that an event e_{12} inherits all properties of an event e_1. An event table can be further reduced through elimination of events that occur in every instance of their parent events and such that they have only one parent event.

TABLE I: AN EVENT TABLE

Event	Time units	Signature
e_1	$\{t_1\}$	(e_{11}, e^2_{12})
e_2	$\{t_2\}$	(e_{11})
e_2	$\{t_3\}$	(e_{11}, e_{13})
e_{11}	$\{t_1, t_3, t_4\}$	Ø
e_{12}	$\{t_2, t_3\}$	(e_{121})
e_{121}	$\{t_2, t_3\}$	Ø
e_{13}	$\{t_5\}$	Ø

In the example above, the events e_{12}, e_{121} occur in all instances of one parent event and because of that they can be removed from the event table given in a Table I. An event e_{11} has two parents (e_1, e_2^2) and it cannot be removed. An event e_{13} cannot be removed because it has one parent (e_2) and does not occur in all instances of its single parent because an event e_2 has two instances in a nested log. A reduced event table is given in a Table II.

TABLE II: A REDUCED EVENT TABLE

Event	Time units	Signature
e_1	$\{t_1\}$	(e_{11})
e_2	$\{t_2\}$	(e_{11})
e_2	$\{t_3\}$	(e_{11}, e_{13})
e_{11}	$\{t_1, t_2, t_3\}$	Ø
e_{13}	$\{t_5\}$	Ø

IV. WORKLOAD TRACE

A model of *workload trace* described in this section is based on the model developed earlier for the audit trails in database systems [19]. Let $|U|$ denotes the total number of time units in U and let $U[n]$ denotes the n-th time unit in U where n changes from *1* to $|U|$. A *workload trace of an event e* is a sequence W_e of $|U|$ multisets of events such that $W_e[i] = (e^k{}_i)$ or $W_e[i]$ = Ø for i=1,...,$|U|$ and $k_i \geq 1$ equal to the total number of times an event e has been processed in the i-th time unit $U[i]$.

If an event e occurred in a time unit t then it may happen that a period of time when $e{:}t$ is processed overlaps on one or more time units t +1, t +2,... . In this work we consider only periodic processing of events determined by the time units where each event has started and we ignore its processing overlaps on the successive time units. In order to eliminate an obviously incorrect classification of an event over time units when the event starts at the very end of time unit, e.g. during the fraction of seconds just before the end of time unit, and almost all of its processing is happens in the successive time units we move such event to the next time unit where the majority of processing has happened.

Let E be a set of all events whose occurrences are recorded in a nested log $L(E)$ over time units U and saved in a reduced event table.

A *workload trace of a nested log* $L(E)$ is denoted by W_L and $W_L[i] = \uplus_{e \in E} W_e[i]$, $\forall i=1,\ldots,|U|$, i.e. it is a multiset union (denoted by $\uplus$) of workload traces of all events included in $L(e)$.

V. Periodic Pattern

In this work we consider periodic patterns consistent with a general model defined as a triple $<C, R, P>$ where: C denotes a *carrier* of a periodic pattern which determines a structure of periodically repeated events, computations, queries, etc., R denotes a *range* of periodic repetitions of a *carrier* measured in time units, for example from time unit t_i to a time unit t_j with a possible extension by n time units, P denotes a *periodicity* that determines when the next periodic repetition of a *carrier* may happen, for example after p time units from the latest occurrence of a *carrier* with possible delay by k time units.

For example, [19] defines C as a nonempty sequence of multisets of syntax trees of relational algebra expressions implementing SQL statements, R is a pair of the ordinal numbers of time unit where the repetitions of a carrier start and end, and P is a total number of time units between two adjacent repetition plus one. In [2] C is defined as a multiset e^k of an event e, R is defined as a pair of numbers $f{:}t$ that determine the first and the last repetition of a carrier and P is defined as a pair of numbers $n{:}x$ that determine the minimal and maximum distance between any two adjacent repetitions of a carrier.

In this work we consider a simplified version of periodicity model used in [2]. A *periodic pattern* is defined as a triple $<C, f{:}t, p>$ where a *carrier C* is a nonempty sequence of at least one nonempty multisets of events, a *range* $f{:}t$, is a pair of *from* and *total* natural numbers, and *periodicity* p is a natural number. Additionally the values of $f{:}t$ and p satisfy the conditions $f \geq 1$ and $t \geq 1$ and $p \geq 0$ and $f + (t-1) * p + |C| - 1 \leq |U|$ and if $t = 1$ then $p = 0$.

Interpretation of periodic patterns is formalized through the following sequence of definitions. Let C be a sequence of multisets where $|C| \leq n$. A trace of C spanning over n multisets and starting at a time unit f where $f + |C| - 1 \leq n$ is denoted by $tr(C, f, n)$ and it is defined as sequence of f-1 empty multisets followed by a sequence of multisets C and followed by n-(f +1)-$|C|$ empty multisets. For example, $trace(e_1e_2^2,3,5)$ is a sequence of multisets $\emptyset\, \emptyset\, e_1e_2^2 \emptyset$.

Consider a periodic pattern $<C, f{:}t, p>$. A trace of the periodic pattern over n time units where $f + t*p - 1 + |C| \leq n$ is denoted by $TR(<C,f{:}t,p>,n)$ and it is defined as a union $tr(C, f, n) \uplus_{1..n} tr(C, f+p, n) \uplus_{1..n} \ldots \uplus_{1..n} tr(C, f+(t-1)*p, n)$ where $\uplus_{1..n}$ denotes multiset union of the respective elements of traces from 1 to n. For example, a trace of periodic pattern $<e_1e_2^2$, 2:2, 1> over 5 time units is the following union of sequences of multisets $\emptyset\, e_1e_2^2 \emptyset\, \emptyset \quad \uplus_{1..5} \quad \emptyset\, \emptyset\, e_1e_2^2 \emptyset = \emptyset\, e_1(e_1,e_2^2)e_2^2\emptyset$

In a special case when a value of $t = 1$ then a value of parameter p must be equal to zero, for example, a trace of periodic pattern $<e_1e_2^2$, 2:1, 0 > over 5 time units is equal to $\emptyset\, e_1e_2^2 \emptyset\, \emptyset$.

In another special case when a value of parameter $p = 0$ and a value of parameter $t \neq 0$ a trace of periodic pattern $<e_1e_2^2$, 2:2, 0 > over 5 time units is equal to $\emptyset\, e_1^2e_2^4 \emptyset\, \emptyset$.

Let W_L be a workload trace of a nested log L. We say, that a periodic pattern $<C, f{:}t, p>$ *is valid in a workload histogram* W_L recorded over time units U if $TR(<C,f{:}t,p>, |U|)[i] \subseteq W_L[i]$ for $i=1,\ldots,|U|$, in the other words if every element of trace of the pattern over $|U|$ time units is included in the respective element of a workload W_L.

For example, a periodic pattern $<e_1e_2^2$, 2:2, 1> is valid in a workload trace $e_1^3e_1(e_1^2,e_2^2)e_2^2\emptyset$ because every element of its trace $\emptyset\, e_1(e_1,e_2^2)e_2^2\emptyset$ is included in the respective element of the workload trace.

VI. Derivation Rules

The derivation rules presented below allow for the logical inference of new periodic patterns from a set of periodic patterns already identified in a workload trace W_L.

Rule 0 (*Normalization*) If a periodic pattern $<C, f{:}t, p>$ is valid in a workload W_L then a periodic pattern $<C', f'{:}t, p>$ such that C' is obtained from C through elimination of all i leading empty multisets and all trailing empty multisets and such that $f' = f + i$ is valid in W_L. *Normalization rule* allows for elimination of leading and/or trailing empty multisets from a carrier of a periodic pattern.

Rule 1 (*Synthesis*) If the periodic patterns $<C, f_i{:}1, 0>$ *and* $<C, f_j{:}1, 0>$ such that $f_i \leq f_j$ are valid in W_L then a periodic pattern $<C, f_i{:}2, f_j - f_i>$ is valid in W_L. For example, if the periodic patterns $<e_1e_2^2$, 2:1, 0> and $<e_1e_2^2$, 5:1, 0> are valid in WL then a periodic pattern $<e_1e_2^2$, 2:2, 3> is valid in W_L.

Rule 2 (*Extension*) If the periodic patterns $<C, f_i{:}t_i, p>$ and $<C, f_j{:}t_j, p>$ are valid in a workload W_L and $f_j = f_i + p$ then a periodic pattern $<C, f_i{:}t_i+t_j, p>$ is valid in W_L. For example, if a periodic pattern $<e_1e_2^2$, 2:2, 3> obtained from application of the *synthesis* rule in the previous example and periodic pattern $<e_1e_2^2$, 7:1, 0> are valid in a workload trace W_L then a periodic pattern $<e_1e_2^2$, 1:3, 3> is valid in W_L.

Rule 3 (*Restriction*) If a periodic pattern $<C, f{:}t, p>$ is valid in a workload W_L then a periodic pattern $<C, f'{:}t', p>$ such that $f' = f+(n-1)*p$ for $n = 1,\ldots,t$ *and* t' such that $f' + (t'-1)*p \leq |U|$ is valid in W_L. For example, if a periodic pattern $<e_1e_2^2$, 2:4, 3 > is valid in W_L then a periodic pattern $<e_1e_2^2$, 5:2, 3 > obtained through the elimination of the first and the last cycle is valid in W_L.

Rule 4 (*Decomposition*) If a periodic pattern $<C, f{:}t, p>$ is valid in a workload W_L then a periodic pattern $<C', f'{:}t, p>$ such that a carrier C' is obtained from a carrier C by elimination of any multiset from any element of a carrier C and if there are any leading or trailing empty multisets in C' elimination of all i leading empty multisets and all trailing empty multisets from C' is valid in W_L. For example, if a periodic pattern $<e_1e_2^2$, 2:5, 1> is valid in W_L then a periodic pattern $<e_2$, 3:5, 1> obtained through the elimination e_1 from the first element and e_2 from the second element of a carrier $e_1e_2^2$ is valid in W_L.

Rule 5 (*Composition*) If the periodic patterns $<C_i, f_i{:}t, p >$ and $<C_j, f_j{:}t, p>$ are valid in a workload W_L and $f_i \leq f_j$ then a periodic pattern $<C_k, f_i{:}t, p>$ where $C_k = tr(C_i, 1, f_j - f_i + |C_j|) \uplus_{1..p} tr(C_j, f_j - f_i, f_j - f_i + |C_j|)$ *is valid in* W_L. For example, if the periodic patterns $<e_1e_2^2, 1{:}3, 4>$ and periodic pattern $<e_1, 4{:}3, 4>$ are valid in a workload trace W_L then a periodic pattern $<e_1e_2^2 \emptyset\, e_1, 1{:}3, 4>$ is valid in W_L.

Rule 6 (*Grouping*) If a periodic pattern $<C, f{:}t, p>$ is valid in a workload W_L then for any $n=1,\dots,t$ such that $t \bmod n = 0$ a periodic pattern $<C', f{:}t/n, p*n>$ such that a carrier $C' = TR(<C, f{:}t, p>, (n-1)*p+|C|))$ is valid in W_L. For example, if a periodic pattern $<e_1e_2^2, 1{:}4, 3>$ is valid in W_L then a periodic pattern $<e_1e_2^2 \emptyset\, e_1e_2^2, 1{:}2, 6>$ is valid in W_L.

Grouping and *composition* derivation rules can be used to combine two periodic patterns with different carriers, ranges and periodicity. As a simple example consider the periodic patterns $<e_1e_2^2, 1{:}4, 3>$ and $<e_3, 3{:}6, 2>$. To apply the *composition* rule the parameters t and p must be the same. To find a common p we find the *least common multiplier* of the periods of the patterns, i.e. $lcm(2,3) = 6$. Then, we apply the *grouping* rule to transform a pattern $<e_1e_2^2, 1{:}4, 3>$ into $<e_1e_2^2 \emptyset\, e_1e_2^2, 1{:}2, 6>$ and pattern $<e_3, 3{:}6, 2>$ *into* $<e_3 \emptyset\, e_3 \emptyset\, e_3, 3{:}2, 6>$. Now, application of the composition rule provides a periodic pattern $<e_1e_2^2e_3e_1(e_2^2, e_3) \emptyset\, e_3, 1{:}2, 6>$.

VII. Discovering Periodic Patterns

Let P be a set of periodic patterns and let R be a set of derivation rules presented in the previous section. We denote by $P \rightarrow_R P'$ a fact that a set of periodic patterns P' can be derived by from a set of patterns P using a set of derivation rules R presented in the previous section. We say that the sets of periodic patterns P_i and P_j are equivalent if $P_i \rightarrow_T P_j$ and $P_j \rightarrow_T P_i$. Discovering the complex periodic patterns is performed through application of the following sequence of algorithms.

A. Algorithm 1

A periodic pattern $<C, f{:}t, p>$ is called as an *elementary periodic pattern* when $C=e$ and $p=1$. The following algorithm finds the elementary periodic patterns $<e, f{:}t, 1>$ in a workload trace W_L for a given event e.

1) Make a multiset of elementary periodic patterns $P(e)$ empty.
2) Iterate over the elements of a workload trace W_L from the first to the last element. Set the first available element in W_L *to* 1.
   - Find the smallest index greater or equal to an index of the first available element in W_L such that $e \subseteq W_L[f]$ and record in t the total number of adjacent elements in a workload histogram such that each element includes e.
   - Append a periodic pattern $<e, f{:}t, 1>$ to a multiset $P(e)$.
   - Modify the entries in a workload trace W_L in the following way:

$W_L[f] := W_L[f]-e$, $W_L[f+1] := W_L[f+1]-e, \dots, W_L[f+t-1]: = W_L[f+t-1]-e$. Such modification is needed to eliminate an impact of a periodic pattern $<e, f{:}t, 1>$ on a workload trace W_L. Next, we set the first available element in W_L *to* $k=f+t$ and if its greater than $|W_L|$ move to step (3), otherwise continue step (2) from (2.1).

3) If at least one new pattern has been added in the latest pass through W_L then move to step (2), otherwise quit the algorithm.

Algorithm 1 is repeated for all events in $\{e_1,\dots,e_n\}$ until no new elementary patterns can be found. As the result, we obtain the multisets of periodic patterns $P(e_1),\dots,P(e_n)$. In a special case it is possible to consider an entire workload trace W_L as a carrier of a periodic pattern $<W_L, 1{:}1, 0>$. Then, Algorithm 1 is a simple application of the *decomposition* rule to the pattern and later on the synthesis and extension rules to create the patterns where $p = 1$.

B. Algorithm 2

The next algorithm applies a *composition* rule to the elementary periodic patterns separately in each $P(e_1),\dots,P(e_n)$ to create new patterns.

1) Find all pairs of periodic patterns $<e^k{}_i, f{:}t, 1>$ and $<e^k{}_j, f{:}t, 1>$ *in* $P(e)$ *and* replace each pair with a periodic pattern $<e^{k_i+k_j}, f{:}t, 1>$.
2) Repeat step (1) until no new periodic patterns can be created.

Algorithm 2 is repeated for all sets of periodic patterns $P(e_1),\dots,P(e_n)$.

C. Algorithm 3

The next algorithm applies the *synthesis* rule to the periodic patterns separately in each $P(e_1),\dots,P(e_n)$ obtained from the previous step to create new patterns.

1) Find a sequence of at least two periodic patterns $<e^k, f_1{:}t, 1>, \dots, <e^k, f_n{:}t, 1>$ in $P(e)$ such that $f_{i+1}-f_i$ is the same for $i=1,\dots,n-1$.

If a sequence cannot be found then quit the algorithm.

2) Apply the *grouping* rule to transform each pattern in the sequence into $<e^k \dots e^k, f_1{:}1, 0>, \dots, <e^k \dots e^k, f_n{:}1, 0>$ where a multiset e^k in a carrier are repeated t times.
3) Apply the *synthesis* and extension rules to transform the sequence into a periodic pattern $<e^k \dots e^k, f_1{:}t, f_j-f_i>$ and replace the sequence with a derived pattern.
4) Repeat the steps (2) and (3) until no new periodic patterns can be created.

Algorithm 3 is repeated for all sets of periodic patterns $P(e_1),\dots,P(e_n)$.

D. Algorithm 4

Finally, the last algorithm applies the *composition* rule to all periodic patterns from a multiset $P(E)=P(e_1) \uplus \dots \uplus P(e_n)$.

1) Find a pair of periodic patterns $<C_i, f_i{:}t, p_i>$ and $<C_j, f_j{:}t, p_j>$ included in $P(E)$ and $f_i \leq f_j$ and such that the patterns overlap more than a given overlap threshold o_{max} determined as $f_j - f_i \leq o_{max}$.

Application of the *restriction* rule may be required to adjust the total number of cycles in the patterns to a common value t.

The rule is applied only if one of the patterns returned is shorter than a given percentage of the total number of cycles of the original pattern, e.g. it consist of less than 20 percent of the total number of cycles of the original pattern.

2) Apply the *grouping* rule to each element of the pair such the derived patterns have a common value of parameter p, i.e, $<C'_i, f_i{:}t', p>$ and $<C'_j, f_j{:}t', p>$.
3) Apply the composition rule to the patterns $<C'_i, f_i{:}t', p>$ and $<C'_j, f_j{:}t', p>$ to create a complex pattern $<C_k, f_i{:}t',$

p>.

4) Replace the original pair of patterns with a pattern obtained in the previous step.
5) Repeat the steps above until no new periodic patterns can be created.

The following simple example shows how the derivation rules implemented with each algorithm above transform the periodic patterns. We start from a workload trace $W_L = e_1^2(e_1,e_2)e_1^2(e_1,e_2)e_1$.

Algorithm 1 transforms a pattern $<W_L, 1{:}1, 0>$ into the following six patterns $p_1=<e_1, 1{:}5, 1>$, $p_2=<e_1, 1{:}1, 0>$, $p_3=<e_1, 3{:}1, 0>$, $p_4=<e_1, 5{:}1, 0>$, $p_5=<e_2, 2{:}1, 0>$, and $p_6=<e_2, 4{:}1, 0>$.

Algorithm 2 does not transform any periodic patterns.

Algorithm 3 transforms the patterns $p_2,\dots,p_6$ into the patterns $p_7 = <e_1, 1{:}2, 2>$ and $p_8 = <e_2, 2{:}2, 2>$.

Algorithm 4 uses the *composition* rule to transform p_7 and p_8 into $p_{11} = <e_1e_2, 1{:}2, 2>$.

Then, it uses the *restriction* rule to transform a pattern p_1 into the patterns $p_9 = <e_1, 1{:}4, 1>$ and $p_{10} = <e_1, 5{:}1, 0>$.

The *grouping* rule is used to transform p_9 into $p_{12} = <e_1e_1, 1{:}2, 2>$.

Finally, the *composition* rule is used again to transform p_{11} and p_{12} into the result $p_{13} = <e_1^2(e_1,e_2), 1{:}2, 2>$.

At the end a set of patterns p_{10}, p_{13} is equivalent to a workload trace W_L.

VIII. Experiment

The main objective of the experiments was to estimate the quality of the proposed system of derivation rules and strategy of their application as a sequence of algorithms transforming the periodic patterns. As a measure of quality we used a ratio of the total number of single cycle periodic patterns obtained after all derivations to the total number of time units in a workload trace.

As a sample nested log of events, we used a transformed audit trail obtained from the monitoring of a synthetic database load generator running against "of-the shelf" commercial relational database management system. An audit trail is a sequence of SQL statements with associated processing timestamps. The sequence becomes the topmost level of a nested log. At the lower levels each SQL statement is transformed with `EXPLAIN PLAN` statement into a syntax tree of an extended relational algebra expression. The operations of the expression form the events in a nested log. At the lowest level in a nested log the operations of *read/write data block* are associated with the timestamps obtained from SQL trace of the statements included in an audit trail.

The database load generator created a sequence of `SELECT` statements that access a sample TPC H benchmark relational database. The stream was audited and audit trail consisting of `SELECT` statements and respective timestamps was saved. At period of time over which an audit trail was collected was divided into a given number of adjacent time units U and `SELECT` statements included in the audit were assigned to the time units in U. In a perfect case there should be no single cycle periodic patterns after all derivations. It means that the algorithms implementing the derivations rules and transformation strategies are able to attach every entry in a workload trace to one of periodically performed actions.

To obtain an input data set consistent with a model of nested log of events `EXPLAIN PLAN` statement was applied to each `SELECT` statement included in the trail to get the syntax trees of extended relational algebra expressions implementing the statements together with information about their timestamps. A root of each syntax tree represented the topmost event, the nodes with the operations of extended relational algebra represented the nested events, and the leaf level nodes represented elementary events with empty logs. The syntax trees were used to create event tree table and later on reduced events tree table in a way described earlier in the paper.

Application of synthetic workload generator allowed us to generate the periodic processing of database tasks with the earlier determined parameters such that the trace of periodic processing could be easily compared with the results of derivations. The main component of a single instance of the generator was a process that iteratively submitted for processing a given sequence of `SELECT` statements in a given period of time. The process could be nested such practically any periodic processing could be obtained in Unix environment.

All software was implemented in SQL embedded in a host language of the database management system used. The software was parameterized in a number of dimensions. First, a period of time over which an audit is performed could be divided into a given number of disjoint and adjacent time units with an earlier determined length of each time unit. A synthetic workload can be easily reconfigured through addition and/or removal of Unix shell scripts running periodically processed SQL scripts. The minimal total number of elements in the sequences of periodic patterns processed by Algorithm 3 was parameterized to avoid the derivations of periodic patterns with a low number of cycles and long carriers. Finally, an overlap parameter that determines the maximum distance between the first cycles of the composed periodic pattern was enforced in implementation of Algorithm 4 to eliminate the compositions of periodic patterns whose first cycles are not close enough.

The experiments performed on the synthetically generated workloads proved, that on average it is possible to reduce the initial number of elementary periodic patterns obtained directly from a workload trace to total 10% of the total number of elementary patterns. The total number of "leftovers", i.e. executions of syntax trees not assigned to any pattern is not larger than 5% of the total size of workload trace. Application of the composition rule allows for creation of complex periodic patterns, however, frequent application of compositions increases the length of a carrier and in the same moment reduces the total number of cycles in the derived patterns.

IX. Summary and Future work

This works shows that it is possible to derive the complex periodic patterns of events from information recorded in the nested event logs and from elementary periodic patterns. A formal model of nested logs of events aggregates information extracted from the application traces, event logs and dynamic profiles. Then, a nested log of events is transformed into a workload trace that binds the events with a predefined set of time units. A system of derivation rules is used to create the complex periodic patterns from the elementary ones in the following way. First, a sequence of algorithms applies the derivations rules to transform a workload trace into a collection of elementary periodic patterns. Next, the derivation rules are used to transform the elementary periodic patterns into the complex periodic patterns. An experiment that processes information included in the database audit trails is verifies the concept and it estimates the quality of discovery process implemented as a sequence of derivations.

The approach presented in the paper tries to reduce the complexity of search for periodic patterns through discovery of single event patterns and later on creation of more sophisticated patterns through application of derivation rules. The important aspects of such approach are logical correctness and completeness of a given system of derivation rules. Correctness of the derivation rules can be easily proved directly from a definition of validity of periodic pattern in a workload trace. We argue that the system of rules is complete for an interpretation of periodic pattern described in the paper because if a workload trace contains a pre-specified periodic pattern then decomposition of workload trace into elementary patterns and later on application of the rules always restores the original pattern.

A formal model of periodic patterns considered in the paper assumes that all cycles of a pattern are always processed with the same periodicity and none of them fails. In the reality, it may happen that a cycle is not processed due to some random reasons or it is delayed such that it happens in one of the following time units. An interesting research objective is a set of rules that can further enhance the periodic patterns obtained from the algorithms described in the paper, for example the rules which can be used to derive complex and imperfect periodic patterns. Such system of derivation rules should be able to detect the cases when some cycles are missing or delayed to another time unit.

Another interesting extension is the best choice of time units over the workload traces. At the moment, a completely arbitrarily selection of time units may result with either too fine or to coarse granulation of time and in a consequence it may distort the periodic patterns. Too long time units may result with the continuous periodic patterns where every element of workload histogram is included in a pattern. Too short time units may provide periodic patterns with low level of quality indicators such as regularity and density. A mechanism is needed to find the most appropriate granulation of time for the parameters of a given event log.

References

[1] N. Bruno, *Automated Physical Database Design and Tuning*, CRC Press Taylor and Francis Group, 2011.

[2] J. Getta, M. Zimniak, and W. Benn, "Mining periodic patterns from nested event logs," in *Proc. the 14th IEEE International Conference on Computer and Information Technology, CIT 2014*, 2014, pp. 160–167.

[3] H. Agrawal and J. Horgan, "Dynamic program slicing," presented at the SIGPLAN Conference on Programming Language Design and Implementation, 1990.

[4] T. Ball and J. R. Larus, "Efficient path profiling," presented at the IEEE/ACM International Symposium on Microarchitecture, 1996.

[5] G. Ammons and J. R. Larus, "Improving data flow analysis with path profiles," presented at the SIGPLAN Conference on Programming Language Design and Implementation, 1998.

[6] X. Zhang, R. Gupta, and Y. Zhang, "Precise dynamic slicing algorithms," presented at the IEEE/ACM International Conference on Software Engineering, 2003.

[7] X. Zhang and R. Gupta, "Matching execution histories of program version," presented at the Joint 10th European Software Engineering Conference and 13th SIGSOFT Symposium on the Foundations of Software Engineering, 2005.

[8] *ACM Trans. Archit. Code Optim.*, vol. 2, no. 3, pp. 301–334, 2005.

[9] E. Siever, A. Weber, S. Figgins, R. Love, and A. Robbins, *Linux in a Nutshell*, O'Reilly, 2005.

[10] B. Ozden, S. Ramaswamy, and A. Silberschatz, "Cyclic association rules," in *Proc. the Fourteenth International Conference on Data Engineering*, 1998, pp. 412–421.

[11] H. Mannila, H. Toivonen, and A. I. Verkamo, "Discovery of frequent episodes in event sequences," *Data Mining and Knowledge Discovery*, vol. 1, pp. 259–289, 1997.

[12] M. Wojciechowski, "Discovering frequent episodes in sequences of complex events," in *Proc. Enlarged Fourth East-European Conference on Advances in Databases and Information Systems (ADBIS-DASFAA)*, 2000, pp. 205–214.

[13] F. Rasheeed, M. Alshalalfa, and R. Alhajj, "Efficient periodicity mining in time series databases using suffix trees," *IEEE Transactions on Knowledge and Data Engineering*, vol. 23, no. 1, pp. 79–94, 2011.

[14] K.-Y. Huang and C.-H. Chang, "SMCA: A general model for mining asynchronous periodic patterns in temporal databases."

[15] J. Yang, W. Wang, and P. S. Yu, "Mining asynchronous periodic patterns in time series data," *IEEE Transactions on Knowledge and Data Engineering*, vol. 15, no. 3, pp. 613–628, Mar. 2003.

[16] J.-S. Yeh, S.-C. Lin, and S.-C. Hu, "Novel algorithms for asynchronous periodic pattern mining based on 2-d linked list," *International Journal of Database Theory and Application*, vol. 5, no. 4, pp. 33–43, 2012.

[17] S. Laxman and P. S. Sastry, "A survey of temporal data mining," *Sadhana, Academy Proceedings in Engineering Sciences*, vol. 31, no. 2, pp. 173–198, 2006.

[18] J. F. Roddick and M. Spiliopoulou, "A survey of temporal knowledge discovery paradigms and methods," *IEEE Transactions on Knowledge and Data Engineering*, vol. 14, pp. 750–767, 2002.

[19] M. Zimniak, J. Getta, and W. Benn, "Deriving composite periodic patterns from database audit trails," in *Proc. the 6th Asian Conference on Intelligent Information and Database Systems*, 2014, pp. 310–321.

[20] D. A. Simovici and C. Djeraba, *Mathematical Tools for Data Mining: Set Theory, Partial Orders, Combinatorics*, Springer, 2008.

Janusz R. Getta received the BSc and MSc degrees in computer science in 1977 from the University of Technology, Warsaw, Poland. He received the PhD degree in computer science in 1983 from the University of Technology, Warsaw. From 1980 to 1986, he was employed as a researcher at the Institute for Scientific, Technical, and Economic Information in Warsaw. From 1986 to 1990 he worked as an associate professor at the University of Kuwait. In 1991 he joined the Department of Computer Science at the University of Wollongong, Wollongong, Australia. He has been working in the field of database for many years. His recent research interests include database performance tuning, data integration, data stream processing, physical database design, and data mining in relational databases.

Marcin Zimniak received his MSc in mathametics from Faculty of Mathematics and Computer Science, Nicolaus Copernicus University, Toruń, Poland in 2001. Currently he is working with the Faculty of Computer Science, TU Chemnitz, Chemnitz, Germany where he is completing his PhD. His relevant research interests include periodic patterns theory, discovering of periodic patterns, efficient data mining algorithm.