Vietnam J Comput Sci (2015) 2:201–211
DOI 10.1007/s40595-015-0042-0

REGULAR PAPER

Predicting database workloads through mining periodic patterns in database audit trails

Marcin Zimniak[1] · **Janusz R. Getta**[2] · **Wolfgang Benn**[1]

Received: 15 June 2014 / Accepted: 6 March 2015 / Published online: 18 April 2015

Abstract Information about the periodic changes of intensity and structure of database workloads plays an important role in performance tuning of functional components of database systems. Discovering the patterns in workload information, such as audit trails, traces of user applications, and sequences of dynamic performance views, is a complex and time-consuming task. This work investigates a new approach to analysis of information included in the database audit trails. In particular, it describes the transformations of information included in the audit trails into a format that can be used for discovering the periodic patterns in the fluctuations of database workloads. It presents an algorithm that finds elementary periodic patterns through nested iterations over a four-dimensional space of execution plans of SQL statements and positional parameters of the patterns. It proposes a collection of composition rules for the derivations of complex periodic patterns from the elementary and other complex patterns and it shows how to use such rules to predict the future workload levels.

Keywords Periodic pattern · Audit trail · Automated performance tuning · Online database design

✉ Janusz R. Getta
jrg@uow.edu.au

Marcin Zimniak
marcin.zimniak@cs.tu-chemnitz.de

Wolfgang Benn
benn@cs.tu-chemnitz.de

[1] Faculty of Computer Science, TU Chemnitz, Chemnitz, Germany

[2] School of Computer Science and Software Engineering, University of Wollongong, Wollongong, Australia

1 Introduction

It is a well-known fact that database workloads periodically change in time. The workload oscillations are caused by the recurrent invocations of database applications that access and change data on behalf of database users operating the typical real-world processes, like for example customers accessing bank accounts, stock brokers performing financial operations, students enrolling course, etc. Discovering the patterns in a continuously changing database workload may to a large extent allow for anticipation of the future operations on data containers. Information about the future levels of workload can be used to automate performance tuning and to improve online database design [6]. In particular, estimation of future query execution time may have an important impact on query scheduling and monitoring of query processing [3,4,16,17]. In a typical scenario, information about the periods of low database workload and about the data containers used by the applications at the periods of higher workload levels enables appropriate restructuring of data containers to speed up the future processing. For example, it is possible to create indexes, to pin data containers in a data buffer cache, to partition data containers, to create materialized views, etc.

Information about the structures and characteristics of the past database workloads can be collected from a database management system in a form of audit trails, traces from processing of SQL statements, sequences of snapshots of internal systems states, dynamic performance views, etc. Due to a large number of concurrently processed user applications, a structure of workload trace is very complex and irregular with no clearly visible patterns in the ways how data containers are accessed and how database operations are processed. This is why ad hoc discovery of complex periodic patterns in a database workload is a difficult and time-consuming task. On the other hand, many of user appli-

cations are processed in the regular ways because the same applications implement the real-world processes, which due to the business or legal reasons must be periodically performed by the individuals and organizations.

This paper addresses a problem how to discover the complex periodic patterns from information included in the database audit trails. We show how to discover the regularities in processing of elementary and complex database operations in a chaos of hundreds of thousands apparently not related database operations. 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 created through processing of simple components of user applications like for example fragments of query processing plans of `SELECT` statements and later on to "derive" the complex periodic patterns from the elementary ones. The idea is justified by the simplicity of candidate periodic patterns selection process. In a general case a set of complex candidate periodic patterns is hidden due to the several dimensions the patterns are defined in and due to a large size of each dimension. For example, a periodic pattern can be defined in a multidimensional space of syntax trees of generalized relational algebra expressions that implement SQL statements, 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 the large workload histories, a multidimensional space of candidate patterns is still finite, however, in practice it is too large for the efficient generation and verification of complex periodic patterns. An approach proposed in the paper reduces a dimension of complex structures of periodic patterns to elementary patterns based on single events, e.g., processing of a single operation of relational algebra and it also reduces periodicity dimension to fixed and restricted in size periods between the repetitions of events. Discovery of elementary periodic patterns is followed by application 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 a workload history to eliminate a "noise" created by the random processing of accidental applications. The systematic derivations of elementary and complex periodic patterns, applications of such patterns to workload history might at the end lead to a precisely defined structure of database workload.

We consider an environment of a relational database system where the user applications are traced and copied into the anonymized *audit trails*. A selected audit trail contains information about the scopes of user applications and about SQL statements processed by the applications. In the next stage, a statement `EXPLAIN PLAN` is used to translate SQL statements included in the trails within the environments of the relevant relational schemas into the precise specifications of execution plans and the estimations of processing costs. The execution plans are represented as syntax trees of expressions built over the arguments like relational tables, relational and/or materialized views and the operators of an extended relational algebra. The plans are initially stored in a syntax tree table, which is later on analyzed to identify and to remove the replicas of the common subtrees as the common fragments of the plans. Finally, the results are saved in the "compressed" *syntax tree table*. Further reduction of the syntax tree table leads to elimination of subtrees whose frequencies of execution are the same as their parent syntax trees. At the end of this process, we obtain a *reduced syntax tree table*. Next, an audit trail passes through "data cleaning" phase which removes the database operations whose impact on a database workload is minimal. In the same stage, a predefined set of time units is used to create a workload trace of an audit trail as a sequence of multisets of syntax trees processed within each time unit. Then, the workload trace is projected into a collection of traces one for each individual syntax tree included in a reduced syntax tree table. The set of traces is used by an algorithm that discovers elementary periodic patterns. In the final stage, the derivation rules are used to combine the elementary periodic patterns into the composite periodic patterns.

The paper is organized in the following way. The next section reviews the previous works on discovering periodicity in time series and in cyclic association rules. Section 3 defines an environment of relational database and it introduces the concepts of audit trail, time units, syntax tree table, and reduced syntax tree table. In the same section, we present an algorithm, that creates a reduced syntax tree table and reduces it to the smallest set of syntax trees required for the next stage. A concept of periodic pattern in a database workload is defined in Sect. 4. An algorithm for discovering the elementary periodic patterns is explained in Sect. 5. Section 6 present the derivations rules for periodic patterns. Application of the rules to estimation of the future database workload levels is discussed in Sect. 7. Section 9 concludes the paper.

2 Related work

Data mining techniques that inspired the works on periodic patterns came from the works on mining association rules [2] and later on from mining frequent episodes [10] and its extensions on mining complex events [15].

The problem seems to be very similar to a typical periodicity mining in time series [8,12], where analysis is performed on the long sequences of elementary data items discretized into a number of ranges and associated with the timestamps.

In our case, input data are a sequence complex data processing statements, like for example SQL statements and due to its internal structure cannot be treated in the same way as analysis of elementary data elements in time series or genetic sequences. The complex data processing statements form a lattice whose elements are syntax trees of the statements with a partial order determined by an inclusion relationship on syntax trees [14].

The recent approaches, which addressed full periodicity, partial periodicity, perfect and imperfect periodicity [7], and asynchronous periodicity [18] are all based on fixed size and adjacent time units and fixed length of discovered patterns. In our case, the cycles are pretty well determined by the real-world events and because of that it may have variable size and non-contiguous structure.

Our problem is also similar to a problem of mining cyclic association rules [11] where an objective is to find the periodic executions of the largest sets of items that have enough support. A sample cyclic association rule states that two database applications are computed in more or less the same period of time every day just after midnight. In our case, the largest sets of operations do not necessarily mean the highest workload and sometimes a single periodically processed application significantly contributes to a database workload. Additionally, mining of cyclic association rules is not able to discover two or more periodic patterns whose cycles overlap on the same period of time due to the same application processed with different frequencies by two or more users. Invocation of operation on data along the various points in time can be easily described by temporal predicates within a formal scope of temporal programming logic and temporal deductive database systems [1,5]. The reviews of data mining techniques based on analysis of ordered set of operations on data performed by the user applications are available in [9,13]. The model of periodicity considered in this paper is a significant extension of the model proposed in [19] with the new concepts of overlapping periodic patterns and derivations of periodic patterns. A new concept of *support* used in the present work allows to represent the cases where the patterns are randomly weakened due to the fluctuations in processing of database applications.

3 Database processing model

We consider a typical relational database system where the relational model of data is used to represent data containers. Let x be a nonempty set of attribute names later on called as a *relational schema* and let $\mathrm{dom}(a)$ denotes a domain of attribute $a \in x$. A *tuple* t defined over a schema x is a full mapping $t : x \to \cup_{a \in x} \mathrm{dom}(a)$ and such that $\forall a \in x$, $t(a) \in \mathrm{dom}(a)$. A *relational table* r created on a schema x is a set of tuples over a schema x.

Query processor transforms SQL statements submitted by the user applications into the query execution plans formulated as the expressions of extended relational algebra. The operations of extended relational algebra include the implementation-dependent variants of operations of standard relational algebra such as *selection*, *projection*, *join*, *antijoin*, *set operations*, and other operations like *grouping*, *sorting*, and *aggregate functions*. Due to the different implementation techniques, the operations included in the basic system of relational algebra, e.g. *selection* or *join* contribute to an number of different elementary operations depending on their implementations, e.g. *index based selection*, *full scan selection*, *hash based join*, *index based join*, etc.

3.1 Audit trail

SQL statements submitted by the *user applications* $a_1, \ldots, a_n$ are recorded in an *application trace*. A *trace of an application* a_i is a finite sequence of pairs $\langle c_i{:}t_{c_i}, s_{i_1}{:}t_{i_1}, \ldots, s_{i_n}{:}t_{i_n}, d_i{:}t_{d_i} \rangle$ where c_i is a *connect* statement, t_{c_i} is a timestamp when the statement has been processed, each s_{i_j} is SQL statement with a timestamps t_{i_j} attached, and d_i is a *disconnect* statement with its timestamp t_{d_i}. Processing of an application a_i starts from processing of a connect statement c_i, the processing of SQL statements s_{i_j}, and it finally ends with processing of a disconnect statement d_i.

An *audit trail* is a sequence of interleaved trails of user applications. For example, a sequence $\langle c_i{:}t_{c_i}, s_{i_1}{:}t_{i_1}, c_j{:}t_{c_j}, s_{j_1}{:}t_{j_1}, s_{i_2}{:}t_{i_2}, d_i{:}t_{d_i}\ d_j{:}t_{d_j} \rangle$ is a sample audit trail from the processing of applications a_i, and a_j. In practice, SQL statements can be easily extracted from an audit trail and `EXPLAIN PLAN` statement can be used in the contexts of respective user schemas to transform the statements into the syntax trees of query execution plans over a set of operations of extended relational algebra. A complete information about syntax trees obtained from a database audit trail is kept in a *syntax tree table*.

3.2 Syntax tree table

Let s_i and s_j be SQL statements included in an audit trail and let T_{s_i}, T_{s_j} be the syntax trees obtained by application of `EXPLAIN PLAN` statement to the statements recorded in an audit trail. The codes of operations of extended relational algebra are used as the labels of non-leaf nodes and the names of data containers processed by the operations are the labels of leaf nodes in a syntax tree. The operations of selection on the same arguments are considered as identical no matter what the predicates are used in the selections. For example, a selection $\sigma_{a>10}(r)$ is equivalent to a selection $\sigma_{d='\mathrm{sales}'}(r)$. The same applies to the binary operations of join an antijoin, i.e. two identical implementations of joins on the same arguments and with a different join condition

are equivalent because the complexities of computations are exactly the same. For example, $r \bowtie_{r.a=s.b} s$ is equivalent to $r \bowtie_{r.c=s.d} s$.

If there exists a non-leaf node n in a syntax tree T_{s_j} such that a subtree with a root node n is the same as a syntax tree T_{s_i} then we say that a syntax tree T_{s_i} is *included in or equal to* a syntax tree T_{s_j}, and we denote by $T_{s_i} \sqsubseteq T_{s_j}$.

A *syntax tree table* contains a complete and compressed information about the syntax trees of SQL statements extracted from an audit trail. A syntax tree is represented in a syntax tree table only once no matter how many times it is included in the other syntax trees. A syntax tree table is a set of tuples $\langle tree, operation, left, right, workload, timestamps\rangle$ where $tree$ is a unique identifier of a syntax tree, $operation$ is a code of extended relational algebra operation at the root of syntax tree identified by $tree$, $left$ and $right$ are the identifiers of left and right argument of syntax tree identified by $tree$ or the names of relational tables, $workload$ is an average workload required to process a syntax tree, and $timestamps$ is a set of all timestamps when a syntax tree $tree$ was processed by a database system.

We say that a subtree t_{leaf} is a *leaf level subtree* of a syntax tree T_s if $t_{\text{leaf}} \sqsubseteq T_s$ and both arguments of an operation in a root node of t_{leaf} are data containers.

A syntax tree table is created in the following way.

Step 1: In the first step, we create an empty syntax tree table.

Step 2: Next, we iterate over all statements included in an audit trail starting from the first statement in the trail. All connect and disconnect statements are ignored. We collect the next statement s from the trail and we create its syntax tree T_s. If no more statements are available in the trail then the process of creating a syntax tree table stops.

Step 2.1: We iterate from left to right over all leaf level subtrees in T_s. Let t_{leaf} be the next leaf level subtree in T_s.

Step 2.1.1: With the current leaf level subtree t_{leaf}, we search a syntax tree table for a tuple that has a value of $code$ equal to an operation code in the root node of t_{leaf} and $left$ equal to the left argument of t_{leaf} and $right$ equal to the right argument of t_{leaf}.

Step 2.1.2: If a tuple is found then it means that a subtree the same as t_{leaf} has been already recorded in the table and we append a timestamp of t_{leaf} to $timestamps$ in the tuple found.

Step 2.1.3: otherwise, we append a new tuple to a syntax tree table. The new tuple obtains automatically generated identifier $tree$, code of operation in the root of t_{leaf} becomes a value of $code$ in the tuple. The values of left and right arguments of t_{leaf} become the values of $left$ and $right$, and these are either the names of relational tables or the identifiers of syntax trees already recorded in the table. An average workload needed to process the syntax tree t_{leaf} is estimated using a workload needed to process its subtrees an information about an operation in the root of t_{leaf}. Finally, a timestamp of t_{leaf} is appended to a set $timestamps$ in the tuple.

Step 2.1.4: A subtree t_{leaf} is removed from T_s such that a root node of t_{leaf} is replaced either with an identifier $tree$ found in Step 2.1.2 or with a new identifier $tree$ created in a Step 2.1.3.

Step 2.1.5: If T_s still has at least one leaf level subtree then return to Step 2.1 otherwise processing of T_s is completed and we return to a Step 2.

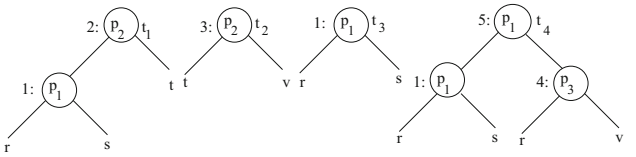

Fig. 1 A sequence of syntax trees

Table 1 A sample syntax tree table

Tree	Operation	Left	Right	Workload	tstamps
1	p_1	r	s	w_1	$\{t_1, t_3, t_4\}$
2	p_2	1	t	w_2	$\{t_1\}$
3	p_2	t	v	w_3	$\{t_2\}$
4	p_3	r	v	w_4	$\{t_4\}$
5	p_1	1	4	w_5	$\{t_4\}$

As a simple example consider a sequence of syntax trees processed at the timestamps t_1, t_2, t_3, and t_4 given in Fig. 1 above where p_1, p_2, and p_3 are the codes of operations. The respective syntax tree table is given in Table 1. The syntax trees 5 and 2 share the common subtree 1. An argument r is shared by the syntax trees 1 and 4. The arguments t, and v are shared by the syntax trees 2 and 3.

3.3 Time units

Let $\langle t_{\text{start}}, t_{\text{end}}\rangle$ be a period of time over which an audit trail is recorded. The period is divided into a contiguous sequence of disjoint and fixed-size *elementary time units* $\langle t_e^{(i)}, \tau_e\rangle$ where $t_e^{(i)}$ for $i = 1, \ldots, n$ is a timestamp when an elementary time unit starts and τ_e is a length of the unit. Elementary time units are distributed over $\langle t_{\text{start}}, t_{\text{end}}\rangle$ such that $t_{\text{start}} = t_e^{(1)}$ and $t_e^{(i+1)} = t_e^{(i)} + \tau_e$ and $t_e^{(n)} + \tau_e = t_{end}$.

A *time unit* is a pair $\langle t, \tau\rangle$ where t is a start point of a unit and τ is a length of the unit. A time unit consists of one or more consecutive elementary time units. A nonempty sequence U of n disjoint time units $\langle t^{(i)}, \tau^{(i)}\rangle$ $i = 1, \ldots, n$

over $\langle t_{\text{start}}, t_{\text{end}}\rangle$ is any sequence of time units that satisfies the following properties: $t_{\text{start}} \leq t^{(1)}$ and $t^{(i)} + \tau^{(i)} \leq t^{(i+1)}$ and $t^{(n)} + \tau_{(n)} \leq t_{\text{end}}$.

As a simple example consider an audit trail that starts on $t_{01:01:2007:0:00\text{a.m.}}$ and ends on $t_{31:01:2007:12:00\text{p.m.}}$. Then, a sequence of disjoint time units 30 min long and usually called as *morning tea time* consists of the following units: $\langle t_{01:01:2007:10:30\text{a.m.}}, 30\rangle$, $\langle t_{02:01:2007:10:30\text{a.m.}}, 30\rangle$, ..., $\langle t_{31:01:2007:10:30\text{a.m.}}, 30\rangle$.

3.4 Workload trace

Let U be a nonempty sequence of n disjoint time units over which an audit trail is recorded and let $|U|$ denotes the total number of time units in U. Then, $U[n]$ denotes the n-th time unit in u where n changes from 1 to $|U|$.

A multiset M is defined as a pair $\langle S, f\rangle$ 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 [14]. In the rest of the paper, we shall denote a multiset $\langle\{T_1, \ldots, T_m\}, f\rangle$ where $f(T_i) = k_i$ for $i = 1, \ldots, m$ as $(T_1^{k_1} \ldots T_m^{k_m})$. We shall denote an empty multiset $\langle\emptyset, f\rangle$ as $\emptyset$ and we shall abbreviate a multiset (T^k) to T^k and T^1 to T.

A *workload trace of a syntax tree* T is a sequence W_T of $|U|$ multisets of syntax trees such that $W_T[i] = \langle\{T\}, f_i\rangle$ and $f_i(T) = |T.timestamp(i)|\ \forall i = 1, \ldots, |U|$, i.e. $f_i(T)$ is equal to the total number of times a syntax tree T was processed in the i-th time unit $U[i]$. A workload trace can be created from information about time units in U and the values in a column $timestamps$ in a syntax tree table.

Let $\mathbf{T}$ be a set of all syntax trees obtained from an audit trail A and recorded in a syntax tree table. A *workload trace of an audit trail* A is denoted by W_A and $W_A[i] = \biguplus_{T\in\mathbf{T}} W_T[i], \forall i = 1, \ldots, |U|$, i.e. it is a sum of workload traces of all syntax trees included in a syntax tree table.

4 Periodic patterns

A *periodic pattern* is a tuple $\langle\mathcal{T}, U, b, p, e\rangle$ where $\mathcal{T}$ is a nonempty sequence of multisets of syntax trees, U is a sequence of disjoint time units that partitions the audit trail into disjoint sequences of SQL statements, $b \geq 1$ is a number of time unit in U where the repetitions of $\mathcal{T}$ start, $p \geq 1$ is the total number of time units after which processing of $\mathcal{T}$ is repeated in every processing cycle, $e > b$ is a number of time unit in U where the processing of $\mathcal{T}$ is performed for the last time. A sequence of multisets $\mathcal{T}$ may contain one or more empty multisets. The positional parameters b, p, and e of a periodic pattern must satisfy a property $(e - b) \bmod p = 0$. A value $c = \frac{e-b}{p} + 1$ is called as the *total number of cycles* in the periodic pattern.

Let $|\mathcal{T}|$ denotes the length of a sequence of multisets $\mathcal{T}$. A *workload trace of a sequence of multisets* $\mathcal{T}$ *starting at the* k-th *time unit*, $1 \leq k \leq |U| - |\mathcal{T}| + 1$ *and spreading over* $|U|$ *time units* is denoted by $W_{\mathcal{T}_k}$ and it is defined as a sequence of multisets $\mathcal{T}$ extended on the left with $k - 1$ empty sets and on the right with $|U| - k - |\mathcal{T}| + 1$ empty multisets. For example, a workload trace of a sequence of multisets $\emptyset TV$ starting at the third time unit and spreading over 9 time units is a sequence of multisets $\emptyset\emptyset\emptyset TV\emptyset\emptyset\emptyset\emptyset$.

Then, a *workload trace of a periodic pattern* $\langle\mathcal{T}, U, b, p, e\rangle$ with the total number of cycles $c = \frac{e-b}{p} + 1$ is a sequence of $|U|$ multisets of syntax trees such that $W_{\mathcal{T}}[i] = W_{\mathcal{T}_b}[i] \uplus W_{\mathcal{T}_{b+p}}[i] \uplus W_{\mathcal{T}_{b+2*p}}[i] \uplus \ldots W_{\mathcal{T}_{b+(c-1)*p}}[i]$ for $i = 1, \ldots, |U|$.

As a simple example, consider a periodic pattern $\langle\emptyset TV, U, 1, 3, 7\rangle$ where processing of a sequence of syntax trees $\emptyset TV$ starts in the time units 1, 4, and 7. Then, the workload traces of $\emptyset TV$ starting at the time units 1, 4, and 7 and spreading over all 9 time units in U are the sequences of multi sets $\emptyset TV\emptyset\emptyset\emptyset\emptyset\emptyset$, $\emptyset\emptyset\emptyset TV\emptyset\emptyset\emptyset$, and $\emptyset\emptyset\emptyset\emptyset\emptyset\emptyset\emptyset TV$. Hence, a workload trace of a periodic pattern $\langle\emptyset TV, U, 1, 3, 7\rangle$ is a sequence of multisets $\emptyset TV\emptyset TV\emptyset TV$. The periodic pattern has three cycles.

We say that *periodic patterns* $\langle\mathcal{T}_i, U_i, b_i, p_i, e_i\rangle$ and $\langle\mathcal{T}_j, U_j, b_j, p_j, e_j\rangle$ *are equivalent* when $U_i = U_j$ and the workload traces of the patterns are the same.

For example, a periodic pattern $\langle\emptyset TV, U, 1, 3, 7\rangle$ is equivalent to a pattern $\langle TV, U, 2, 3, 8\rangle$ which also has three cycles.

When the length of $\mathcal{T}$ in a periodic pattern is longer than a value of parameter p then the cycles of the pattern overlap. For example, $\langle\emptyset TV, U, 1, 1, 3\rangle$ is a periodic pattern where processing of a sequence of syntax trees $\emptyset TV$ starts in the time units 1, 2, and 3 and its workload trace is a sequence of multisets $\emptyset T(VT)(VT)V$. The periodic pattern also has three cycles.

Let $|W_{\mathcal{T}}|$ be the total number of elements in $W_{\mathcal{T}}$. Let v be the total number of elements in $W_{\mathcal{T}}$ such that $W_{\mathcal{T}}[i] \sqsubseteq W_A[b + i - 1]$ for $i = 1, \ldots, e - b + |\mathcal{T}|$ where a symbol $\sqsubseteq$ denotes an inclusion of multisets. Then, we say that a periodic pattern $\langle\mathcal{T}, U, b, p, e\rangle$ is *valid in an audit trail* A *with a support* $0\langle\sigma \leq 1$ if $W_{\mathcal{T}}[1] \sqsubseteq W_A[b]$ and $W_{\mathcal{T}}[e - b + |\mathcal{T}|] \sqsubseteq W_A[e + |\mathcal{T}|]$ and $\sigma \leq v/|W_{\mathcal{T}}|$.

For example, a periodic pattern $\langle(T^2V)\emptyset W, U, 1, 3, 7\rangle$ has a workload trace $(T^2V)\emptyset W(T^2V)\emptyset W(T^2V)\emptyset W$. The pattern is valid in an audit trail A with support $\sigma = 1$ if every element of its workload trace is included in a workload trace W_A from position 1 to position 9.

Let $(T^2VW)\emptyset W^2(TV)\emptyset W^2(T^2V)\emptyset W$ be a workload trace of an audit trail. Then, a periodic pattern $\langle(T^2V)\emptyset W, U, 1, 3, 7\rangle$ is valid in the audit trail with support $\frac{8}{9}$ because a multiset of syntax trees (T^2V) is not included in a multiset (TV) of syntax trees processed by a database system in the fourth time unit.

5 Discovering elementary periodic patterns

We say that a periodic pattern $\langle \mathcal{T}, U, b, p, e\rangle$ is an *elementary periodic pattern* when a sequence of multisets $\mathcal{T}$ consists of one multiset that consists only of k identical elements, i.e. $\mathcal{T} = T^k$. A periodic pattern which is not elementary is called as a *composite periodic pattern*.

In this work, we start from discovering elementary periodic patterns. Then, we show how to create composite periodic patterns using compositions of elementary and other composite periodic patterns. Discovering elementary periodic patterns can be done over a number of dimensions such as syntax trees of all statements included in an audit trail, all possible partitions of audit time into time units in U, all workload levels expressed as multiplicity coefficients in multisets, and the dimensions of positional parameters b, p, and e. In this work, we assume that a sequence of time units U does not change. In the first stage, we perform "data cleaning" of the contents of workload trace W_A. For all multisets included in $W_A[i], i = 1, 2, \ldots |U|$, pre-calculating of *threshold workload* w is proceeded, afterwards it is used to eliminate from a workload trace W_A of an audit trail A created over a fixed sequence of time units U, all multisets T^k such that their cumulative workload is below a mentioned threshold value w for all time units. "Cleaning data process" of an audit trail eliminates SQL statements which do not significantly contribute to an overall workload, like for example a statement which gets the current date and time or a statement that creates a new element of a sequence. More precisely, all multisets T^k are removed from all multisets included in $W_A[i], i = 1, 2, \ldots |U|$ containing T^k if the total contribution to a workload of an multiset $T^{\max(k)}$ in W_A, i.e. the largest number of times a syntax tree T was processed in a time unit i measured as $k * |T.workload[i]|$ is less than the threshold workload w and does not exist at least two multisets (included in some $W_A[i]$, $W_A[j], i \langle\rangle j$) such that each of them contains T and their total workload is greater or equal w. In the other words, "data cleaning" removes from the multisets in a workload trace all elements representing syntax trees whose processing has no significant impact on overall workload and whose processing does not contribute to any periodic patterns.

5.1 Reduced syntax tree table

Let **T** be a set of syntax trees that consists of all syntax trees of statements in a "cleaned" audit trail. Let T_ϵ be an *empty syntax tree* and let T_π be a syntax tree obtained from concatenation of all syntax trees from syntax tree table, which are not included in any other syntax tree. Then, discovering elementary periodic patters is performed over a lattice $\langle \mathbf{T}, \sqsubseteq\rangle$ implemented as a syntax tree table with a minimum T_ϵ and maximum T_π and partial order $\sqsubseteq$ representing inclusion of

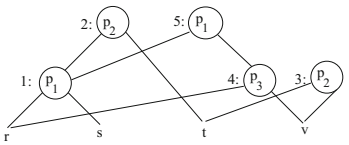

Fig. 2 Visualization of reduced syntax tree table

syntax trees. The following three rules can be used to reduce the total number of iterations over the syntax trees. Let A be an audit trail.

1. If an elementary periodic pattern $\langle T_i^k, U, b, p, e\rangle$ is valid in A then for any syntax tree T_j such that $T_j \sqsubseteq T_i$ and $k' \leq k$ an elementary periodic pattern $\langle T_j^{k'}, U, b, p, e\rangle$ is valid in A.
2. If an elementary periodic pattern $\langle T_i^k, U, b, p, e\rangle$ is valid in A then for any syntax tree T_j such that $T_i \sqsubseteq T_j$ and $\neg\exists T_n \neq T_j$ such that $T_i \sqsubseteq T_n$ an elementary periodic pattern $\langle T_j^{k'}, U, b, p, e\rangle$ where $k' \leq k$ is valid in A.
3. If an elementary periodic pattern $\langle T_i^k, U, b, p, e\rangle$ is not valid in A then for any syntax tree T_j such that $T_i \sqsubseteq T_j$ and $k \leq k'$ an elementary periodic pattern $\langle T_j^{k'}, U, b, p, e\rangle$ is not valid in A.
4. If an elementary periodic pattern $\langle T_i^k, U, b, p, e\rangle$ is not valid in A then for any syntax tree T_j such that $T_j \sqsubseteq T_i$ and $\neg\exists T_n \neq T_i$ such that $T_j \sqsubseteq T_n$ an elementary periodic pattern $\langle T_j^{k'}, U, b, p, e\rangle$ where $k \leq k'$ is not valid in A.

The rules listed above reduce a syntax tree table to a simple table of pairs $\langle tree, timestamps\rangle$ where $tree$ is an identifier of a syntax tree that is supposed to be verified against periodic patterns and $timestamps$ is a set of timestamps when the processing of a syntax tree identified by $tree$ occurred in an audit trail. A *reduced syntax trees table* includes identifiers of all sub-lattices determined by the rules (1)–(4) above. The table contains only information about the syntax trees of the statements from an audit trail and about subtrees shared by two or more syntax trees. For example, a syntax tree table given in Table 1 reduces to a set of pairs $\{\langle 1, \{t_1, t_3, t_4\}\rangle, \langle 2, \{t_1\}\rangle, \langle 3, \{t_2, t_4\}\rangle, \langle 5, \{t_4\}\rangle\}$. There is no need to include a syntax tree 4 because it is a subtree of syntax tree 5 and it is not a subtree of any other syntax tree. Therefore, according to the rules (1) and (4) above any periodic pattern found for a syntax tree 5 will also be valid for a syntax tree 4, see Fig. 2.

5.2 Iterations

Discovering an elementary periodic pattern $\langle T^k, U, b, p, e\rangle$ for a given set of time units U, and a given value of support parameter $0 < \sigma \leq 1$ is performed through the nested iterations over the syntax trees included in a reduced syntax tree

table and the iterations over the positional parameters b, p, and e. At the beginning, all syntax trees in a reduced syntax tree table are marked as "not processed yet" and a set $\mathcal{P}$ of elementary periodic patterns that occur in an audit trial A is set to empty. At each level, the iterations are performed in the following way.

Algorithm 1

Step 1: At the outermost level we pick a syntax tree T from a reduced syntax tree table such that it is not included in any other "not processed yet" syntax tree. If such tree does not exist then the iterations are completed. Otherwise, we create a workload trace W_T for T.

Step 1.1: At the first inner level the iterations are performed over the values of positional parameter b. The parameter b iterates over an increasing sequence of numbers $1, 2, 3, \ldots, |W_T| - 1$. Let b_c be the current value of parameter b. If $W_T[b_c] = \emptyset$ then a value of b_c is increased by one and the same condition is tested again. If no more iterations over the values of parameter b are possible then we move to a Step 1.2 below.

Step 1.1.1: At the next inner level, the iterations are performed over the values of parameter e for a fixed value b_c set at outer level. A parameter e iterates over a decreasing sequence of numbers $|W_T|, |W_T| - 1 \ldots, b_c + 2, b_c + 1$. Let e_c be the current value of parameter e. If $W_T[e_c] = \emptyset$ then we take the next value of parameter e the same condition is tested again. If no more iterations over the values of parameter e are possible we return to Step 1.1.

Step 1.1.1.1: At the lowest level, the iterations are performed over an increasing sequence of values of parameter p such that $(e_c - b_c) \bmod p = 0$ and $b_c + p \langle e_c$. If no more iterations over the values of parameter p are possible we return to Step 1.1.1. Otherwise, we set the current value of parameter p to p_c.

Step 1.1.1.2: Next, we create a candidate elementary periodic pattern $\langle T^{\min(k)}, U, b_c, e_c, p_c \rangle$ where $\min(k)$ denotes the smallest value of k in all instances of element T^k in workload trace W_T in a range $[b_c, e_c]$.

Step 1.1.1.3: We use a trace W_T to check whether the candidate pattern is valid in an audit trail with a given support σ. We compute $c = \frac{e_c - b_c}{p} + 1$ and v as the total number of times $T^{\min(k)}$ is valid in W_T at the positions $b_c, b_c + p_c, \ldots, e_c$. Then, the candidate pattern $\langle T^k, U, b_c, p_c, e_c \rangle$ is valid in an audit trail with a support σ when $T^{\min(k)} \sqsubseteq W_T[b_c]$ and $T^{\min(k)} \sqsubseteq W_T[e_c]$ and $\sigma \leq v/c$. If the candidate pattern is not valid in an audit trail then we return to Step 1.1.1.1 to collect the next value of parameter p.

Step 1.1.1.4: If the candidate pattern is valid in an audit trail then we append $\langle T^{\min(k)}, U, b_c, p_c, e_c \rangle$ to a set $\mathcal{P}$. Then, we modify the entries of trace W_T such that $W_T[i] := W_T[i] - T^{\min(k)}$ for all $\forall i = b_c, b_c + p_c, \ldots, e_c$. Next, we return to Step 1.1.1.1 to collect the next value of parameter p.

Step 1.2: At the end of iterations over the positional parameters, we are left with the single elements in a workload trace W_T, which are not attached to any elementary periodic pattern in $\mathcal{P}$. If there exists a periodic pattern $\langle T^k, U, b, e, p \rangle \in \mathcal{P}$ and an element $W_T[n] = T^m$ such that $n \geq e + p$ and $m \geq k$ then we split the pattern into $\langle T^k, U, b, e - p, p \rangle$ and $\langle T^k, U, e, n, n - e \rangle$ and we modify trace $W_T[n] := W_T[n] - T^{\min(k)}$. Splitting of elementary periodic patterns is repeated until no more single elements in W_T can be used. When finished we mark a syntax tree T as "processed" in a reduced syntax tree table and we return to Step (1) above.

It is important to note, that support of all elementary periodic patterns found by an algorithm described above is equal to 1.

6 Derivation rules for periodic patterns

The derivation rules presented in this section form a basis for the logical reasoning about periodic patterns and for creation of complex patterns through the derivations from the elementary and/or complex ones. We start from a rule that normalizes a sequence of multi sets in a periodic pattern.

Rule 0 (*Normalization*): If a periodic pattern $\langle \mathcal{T}, U, b, p, e \rangle$ is valid in an audit trail A with a support σ then a periodic pattern $\langle \mathcal{T}', U, b', p, e' \rangle$ such that $\mathcal{T}'$ is obtained from $\mathcal{T}$ through elimination of all i leading empty multisets and all trailing empty multisets such that and $b' = b + i$ and $e' = e + i$ is valid in an audit trail A with a support σ.

For example, if a periodic pattern $\langle \emptyset\emptyset(T^2V)\emptyset, U, 1, 3, 7 \rangle$ is valid in an audit trail A then a periodic pattern $\langle (T^2V), U, 3, 3, 9 \rangle$ is also valid in an audit trail A.

Next, we have two simple rules that allow to adjust a length and frequency of periodic pattern.

Rule 1 (*Horizontal split*): If a periodic pattern $\langle \mathcal{T}, U, b, p, e \rangle$ is valid in an audit trail A with a support σ then a periodic pattern $\langle \mathcal{T}, U, b', p, e' \rangle$ such that $b' = b + (m * p)$ and $e' = e - (n * p)$ and $b' \langle e'$ where $m, n \in N^+ \cup \{0\}$ is valid in an audit trail A with a support $\sigma' \geq \sigma$.

For example, if a periodic pattern $\langle (T^2V)\emptyset W, U, 1, 3, 7 \rangle$ is valid in an audit trail A then a periodic pattern $\langle (T^2V)\emptyset W, U, 1, 3, 4 \rangle$ is also valid in an audit trail A.

Rule 2 (*Vertical split*): If a periodic pattern $\langle \mathcal{T}, U, b, p, e \rangle$ is valid in an audit trail A with a support σ then a periodic

pattern $\langle \mathcal{T}, U, b, p', e' \rangle$ such that $p' = n * p$ and $e' \leq e$ and $(e' - b) mod\ p' = 0$ where $n \in N^+ \cup \{0\}$ is valid in an audit trail A with a support $\sigma' \geq \sigma$.

For example, if a periodic pattern $\langle (T^2V)\emptyset W, U, 1, 3, 7 \rangle$ is valid in an audit trail A then a periodic pattern $\langle (T^2V)\emptyset W, U, 1, 6, 7 \rangle$ is also valid in an audit trail A.

The *horizontal* and *vertical split rules* rules can be used to increase a value of support for a periodic pattern at an expense of length and frequency of the pattern and also to adjust the parameters of periodic pattern such that the *extension* and *composition* rules can be applied to build the complex patterns from the simple ones.

Rule 3 (*Decomposition*): If a periodic pattern $\langle \mathcal{T}, U, b, p, e \rangle$ is valid in an audit trail A with a support σ and $\mathcal{T}'[i] \subseteq \mathcal{T}[i]$ for all $i = 1 \ldots |\mathcal{T}|$ then a periodic pattern $\langle \mathcal{T}', U, b, p, e \rangle$ is valid in an audit trail A with a support σ.

For example, if a periodic pattern $\langle (T^2V)\emptyset W, U, 1, 3, 7 \rangle$ is valid in an audit trail A then the periodic patterns $\langle T\emptyset W, U, 1, 3, 7 \rangle$ is also valid in an audit trail A.

The next group of rules shows how to derive the new periodic patterns from information about two periodic patterns valid in an audit trail.

Rule 4 (*Horizontal extension*): Let the periodic patterns $\mathcal{P}_i = \langle \mathcal{T}_i, U, b_i, p_i, e_i \rangle$ and $\mathcal{P}_j = \langle \mathcal{T}_j, U, b_j, p_j, e_j \rangle$ such that $\mathcal{T}_i = \mathcal{T}_j$ and $p_i = p_j$ and $b_j = e_i + p_i$ are valid in an audit trail A with the respective supports σ_i and σ_j. Then a periodic pattern $\mathcal{P}_k = \langle \mathcal{T}_k, U, b_k, p_k, e_k \rangle$ such that $\mathcal{T}_k = \mathcal{T}_i = \mathcal{T}_j$ and $b_k = b_i$ and $p_k = p_i = p_j$ and $e_k = e_j$ is valid in an audit trail A with a support $\sigma_k = \frac{\sigma_i * c_i + \sigma_j * c_j}{c_i + c_j}$ where $c_i = \frac{e_i - b_i}{p_i} + 1$ and $c_j = \frac{e_j - b_j}{p_j} + 1$.

For example, if the periodic patterns $\langle (T^2V)\emptyset W, U, 1, 3, 7 \rangle$ and $\langle (T^2V)\emptyset W, U, 10, 3, 13 \rangle$ are valid in an audit trail A then a periodic patterns $\langle (T^2V)\emptyset W, U, 1, 3, 13 \rangle$ is also valid in an audit trail A.

Rule 5 (*Vertical extension*): Let the periodic patterns $\mathcal{P}_i = \langle \mathcal{T}_i, U, b_i, p_i, e_i \rangle$ and $\mathcal{P}_j = \langle \mathcal{T}_j, U, b_j, p_j, e_j \rangle$ such that $\mathcal{T}_i = \mathcal{T}_j$ and $p_i = p_j$ and $p_i \bmod 2 = 0$ and $b_j = b_i + p_i/2$ and $e_j = e_i + p_i/2$ are valid in an audit trail A with the respective supports σ_i and σ_j. Then a periodic pattern $\mathcal{P}_k = \langle \mathcal{T}_k, U, b_k, p_k, e_k \rangle$ such that $\mathcal{T}_k = \mathcal{T}_i = \mathcal{T}_j$ and $b_k = b_i$ and $p_k = p_i/2$ and $e_k = e_j$ is valid in an audit trail A with a support $\sigma_k = \frac{\sigma_i * c_i + \sigma_j * c_j}{c_i + c_j}$ where $c_i = \frac{e_i - b_i}{p_i} + 1$ and $c_j = \frac{e_j - b_j}{p_j} + 1$.

For example, if the periodic patterns $\langle (T^2V)\emptyset W, U, 1, 4, 5 \rangle$ and $\langle (T^2V)\emptyset W, U, 3, 4, 7 \rangle$ are valid in an audit trail A then a periodic pattern $\langle (T^2V)\emptyset W, U, 1, 2, 7 \rangle$ is also valid in an audit trail A.

Rule 6 (*Composition*): Let the periodic patterns $\mathcal{P}_i = \langle \mathcal{T}_i, U, b_i, p_i, e_i \rangle$ and $\mathcal{P}_j = \langle \mathcal{T}_j, U, b_j, p_j, e_j \rangle$ such that $b_i \geq b_j$ and $\frac{c_i}{c_j} = \frac{p_j}{p_i}$ where $c_i = \frac{e_i - b_i}{p_i} + 1$ and $c_j = \frac{e_j - b_j}{p_j} + 1$ are valid in an audit trail A with the respective supports σ_i and σ_j. Then a periodic pattern $\mathcal{P}_k = \langle \mathcal{T}_k, U, b_k, p_k, e_k \rangle$ such that

(i) $b_k = b_i$ and
(ii) $p_k = \mathrm{LCM}(p_i, p_j)$ and
(iii) $e_k = e_i$ and
(iv) $\mathcal{T}_k = W_{\mathcal{T}_i}^{(k_i)} \uplus W_{\mathcal{T}_j}^{(k_j)}$ where $W_{\mathcal{T}_i}^{(k_i)}$ is a workload trace of the first k_i cycles of $\mathcal{T}_i$ and $W_{\mathcal{T}_j}^{(k_j)}$ is a workload trace of the first k_j cycles of $\mathcal{T}_j$ where $k_i = \frac{c_i * p_i}{\mathrm{LCM}(p_i, p_j)}$ and $k_j = \frac{c_j * p_j}{\mathrm{LCM}(p_i, p_j)}$ where LCM means the Least Common Multiple is valid in an audit trail A with a support σ_k such that $(1 - \sigma_i') + (1 - \sigma_j') \leq \sigma_k \leq \max(\sigma_i', \sigma_j')$ where $\sigma_i' = \frac{\sigma_i * p_i}{\mathrm{LCM}(p_i, p_j)}$ and $\sigma_j' = \frac{\sigma_j * p_j}{\mathrm{LCM}(p_i, p_j)}$.

For example, if the periodic patterns $\langle TV, U, 1, 3, 13 \rangle$ and $\langle W, U, 2, 2, 18 \rangle$ are valid in an audit trail A then a periodic pattern $\langle T(VW)\emptyset(TW)VW, U, 1, 6, 13 \rangle$ is also valid in an audit trail A. To obtain a sequence of multisets for the result of composition the first two cycles from the workload of the first periodic pattern $TV\emptyset TV$ must be combined with the first three cycles of the second periodic pattern $\emptyset W\emptyset W\emptyset W$. As a result, we obtain $T(VW)\emptyset(TW)VW$.

7 Predicting database workloads

Predicting the future database workloads directly from the traces of past workloads does not provide precise results because the traces do not contain information, which workload peaks are caused by the interferences of periodic patterns and which one are simply coincidences of randomly computed complex processing tasks. In this section, we show how the derivations of periodic patterns can be used do create the complex periodic patterns and how to use such patterns for the estimation of future workloads. A process of finding the periodically occurring workload peaks consists of two stages. In the first stage, a set of elementary periodic patterns $\mathcal{P}_e$ and a set of derivation rules described earlier are used to discover a set $\mathcal{P}$ of more complex periodic patterns. In the next stage, a set of time units U over which the periodic patterns are discovered is mapped onto a period of time when the workload levels supposed to be predicted to find workload in the real time in the future.

The derivations of periodic patterns consist of the following steps.

Step 1: A set of elementary periodic patterns $\mathcal{P}_e$ discovered by the Algorithm 1 is partitioned into the disjoint subsets $\mathcal{P}_T$ of periodic patterns such that each $\mathcal{P}_T = \{\langle \mathcal{T}, U, b, p, e \rangle : \mathcal{T} = T^k, k = 1, 2, \ldots\}$. In the other words, partitioning is performed such that all elementary periodic patterns of the same syntax tree are included in the

same partition. Then, each partition is split into the groups $\mathcal{P}_T^f$ depending on a value of parameter p. All periodic patterns such that $p = 1$ are included in a group $\mathcal{P}_T^1$, all periodic patterns such that $p = 2$ are included in a group $\mathcal{P}_T^2$, …all periodic patterns such that $p \bmod f = 0$ where f is a prime number are included in a group $\mathcal{P}_T^f$. At the end of partitioning step, each $\mathcal{P}_T^f$ group contains the elementary periodic patterns of the same syntax tree T and periods p being the multiplicities of the same prime number f.

Step 2: In this step, the longest elementary periodic patterns are created from the patterns included in each group. For each $\mathcal{P}_T^f$ of periodic patterns the followings actions are repeated.
Step 2.1: A group $\mathcal{P}_T^f$ of periodic patterns is analyzed to find the longest sequence of periodic patterns with non-overlapping workload traces and such that a distance between any two workload traces of periodic patterns in a sequence is no longer that a given threshold value.
Step 2.2: All periodic patterns included in a sequence are removed from $\mathcal{P}_T^f$.
Step 2.3: Then, if $k_{\min}$ is the smallest multiplicity of syntax tree T in a sequence than the *decomposition* and *vertical split* rules is used to derive the patterns that have the same multiplicity $k_{\min}$ and the same period p in the sequence.
Step 2.4: A sequence is converted to a single periodic pattern using *horizontal* and *vertical extension* rules.
Step 2.5: A sequence created and the periodic patterns left after application of *decomposition* and *vertical split* rules are returned to a group $\mathcal{P}_T^f$. Then, if it is still possible to create another sequence, a procedure starting from a step 2.1 is repeated. Otherwise, the next group of periodic patterns in considered as an input to step 2.

Step 3: In this step, the *composition* rule is applied to the periodic patterns obtained from the extensions of patterns from all $\mathcal{P}_T^f$ groups to reduce the total number of patterns. We consider all pairs of periodic patterns such that a distance between their start points is no longer than a given threshold value. Then, a *horizontal split* rule is used to adjust the length of periodic patterns in each pair such that *composition* rule can be applied. The results from the compositions, i.e. the composed patterns and the split fragments of patterns not used by composition rules are returned to an input set of periodic patterns and step 3 is repeated. When no more composition can be done the derivations are completed.

In the second stage, a collection of periodic patterns $\mathcal{P}_s$ discovered over a sequence of time units U is used to predict database workloads over a given period of time in the future. Practically, a problem is how to map a sequence of time units U onto a sequence of time units that comprise a period of time in the future. A mapping uses a concept of indexing with the keywords of a sequence of time units in U and a sequence of the respective time units in the future. An *index entry* is a pair $\langle n, U_n \rangle$ where n is a unique name of an entry and U_n is a nonempty subset of time units in U. For example, in a sequence of time units U spreading over a period of 2 weeks where each time units has a length of one day, an index entry $\langle Monday, \{U[1], U[8]\}\rangle$ means giving a name $Monday$ to the first and the eight time unit in U. Another index entry $\langle$*The first day of a month*, $\{U[1]\}\rangle$ names the first time unit in U as *The first day of a month*. Together both entries mean that a sequence of time units U spreads over 2 weeks starting from Monday and also that the first Monday is the first day of a month.

Let I_U be an set of index entries over a sequence of periods of time U. We partition a period of time over which the workloads supposed to be found in to a sequence of time units F with the length of each time unit and distances between time units the same as in U. A sequence of time units F obtains a set of index entries I_F. Finally, U is mapped onto F in a way that maximizes the total number of identical index entries in I_U and I_F. For example, let I_F contains the index entries $\langle Monday, \{F[5], F[12], F[19], F[26]\}\rangle$ and $\langle$*The first day of a month*, $\{F[11]\}\rangle$ and let I_U contains the index entries listed above. Then the best mappings of U are at position $F[12]$ and later on at position $F[26]$ because a location $F[11]$ is closer to $F[12]$ and $F[5]$. The mappings of U onto F and a set of periodic patterns over U determine the future workloads in F.

8 Experiments

The main objective of the experiments was to estimate the quality of algorithms used for identification and derivation of periodic patterns. As a measure of quality, we used a ratio of the total number of entries from workload trace not assigned to any periodic pattern to the total number of entries in a workload trace. The objective of the process was to discover as many periodic patterns as it is possible and to include all entries in a workload trace into at least one periodic pattern. The other measure of quality was a ratio of the total number of periodic patterns obtained after all derivations to the total number of elementary periodic patterns obtained from the first algorithm. The objective was to create as complex periodic patterns as it is possible and to reduce the total number of patterns found. These two measures are somewhat contradictory as application of *extension* and *composition* rules usually creates some "left-overs", i.e. single-entry multisets of syntax trees, which must be returned to a workload trace.

Due to the privacy concerns, we did not use the real audit trails for testing of the algorithms. Instead, we used an audit trail obtained from a synthetic database load generator. It

allowed us to generate the periodic processing of database tasks with the pre-specified parameters such that the created periodic patterns and their traces could be compared with the outcome of the discovery and derivation algorithms. The main component of a synthetic workload generator was a process that iteratively executed a given sequence of SQL statements in a given period of time. For example, a given sequence of `SELECT`, `UPDATE`, `SELECT` statements was iteratively processed in a period of 10 min. The process could be nested such that practically any combination of periodic processing of SQL statement could be obtained in a multi-processing Unix environment.

To get an audit trace, we used as a database server an "off-the-shelf" commercial relational database management system. All software were written in SQL embedded in a host language of the database management system. The clients created the synthetic database workload against a sample TPC H benchmark relational database and an audit trace for a particular single database user was saved in a relational table. When an audit session was completed, the relational table with audit entries was processed such that SQL statements that had no significant impact on performance were filtered out and the remaining statements were used to generate SQL script with `EXPLAIN PLAN` statement together with information about their timestamps. In the next stage, SQL script with `EXPLAIN PLAN` statements and time stamps attached to the statements was used to create a syntax tree table and later on reduced syntax tree table in a way described earlier in the paper. At the end of data generation and preparation stage, a period of time over which an audit trail was collected was divided into a given number adjacent time units U and a relational table with a workload trace W_A was created and loaded with data.

In the first step of periodic pattern discovery, we used a parameterized procedure that found the elementary periodic patterns for the values of parameter $p = 1, 2, 3, \ldots$ that do not exceed a given threshold value. The procedure discovered and saved the elementary periodic patterns starting from the lowest values of parameters p. The experiments conducted with several different strategies of choosing the values of parameter p provided different final sets of periodic patterns depending on the order of the values of parameter p.

In the next step, a set of elementary periodic patterns was passed through the procedures that apply first the *vertical* and *horizontal extension* rules to derive periodic patterns that expand over the longer period of time. Later on, a *composition* rule was applied to the results of extensions to reduce the total number of periodic patterns and to derive the periodic patterns with longer sequences of multisets $\mathcal{T}$. At this stage, the application the *vertical extension* and *composition* rules created the non-elementary periodic patterns. The *horizontal* and *vertical split* rules were used to adjust the parameters of periodic patterns such that *extension* and *composition* rules could be applied. The new periodic patterns obtained from the application of the derivation rules were appended to a set of periodic patterns such that they could be used in the next applications of *extensions* and *compositions*. The periodic patterns created into a single periodic patterns were removed from a set of periodic patterns such that the set always represents the same workload trace. A negative side effect of *split* operation was the "left-overs" which consisted of single processing of syntax trees not covered by the present collection of periodic patterns and saved in a "trash" workload trace. A collection of periodic patterns obtained from the derivations and the contents of "trash" workload trace were equivalent to the original workload trace.

The implemented software is parameterized in a number dimensions. First, a period of time over which an audit is performed can be divided into a given number of disjoint and adjacent time units. A synthetic workload can be very easily reconfigured by adding and/or removing Unix shell scripts running periodically processed SQL scripts. An initial generation on elementary periodic patterns can be controlled by appropriate selection of the values of a parameter p for each pattern and an order in which value of p can be changed. It is possible to enforce a lower limit for the total number of cycles of periodic patterns obtained from the applications of vertical split rule such that periodic patterns with the total number of cycles less than a given threshold value are not created. Finally, it is possible to enforce the upper limit for an offset between the first cycles of the composed periodic pattern. Such parameter eliminates the composition between periodic patterns whose first cycles are not close enough.

The experiments performed on the synthetically generated workloads show 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 $\mathcal{T}$ and in the same moment reduces the total number of cycles. Further application of a rule that find cycles within $\mathcal{T}$ allows to restore the original patterns of the synthetic workloads.

9 Summary and conclusions

Discovering the complex periodic patterns in the database audit trails is a difficult and time-consuming task. An approach investigated in the paper finds the elementary periodic patterns and composes them into the complex ones instead of directly searching for all complex patterns. Direct discovery of complex patterns from the audit trails faces a difficult problem of finding appropriate candidate patterns in

a huge space of theoretically possible sequences of multisets of syntax trees and the values of parameters that determine a scope and periodicity of the patterns. The approach presented in this work is conceptually consistent with many other approaches to discovery of regularities in large data sets. For example, discovery of frequent item sets also starts from the single item candidate item sets, which is later on verified and extended with more items. While the logical inference rules are applied to discard the items which do not satisfy the constraints like support and confidence. An important advantage of the presented approach is the reduction of the total number of times the admit trails must be accessed. The derivation rules which of the periodic patterns satisfy a support constraint and which can be discarded without verification in the smaller components of an audit trail.

A data preparation stage of the process starts from defining the time units and partitioning an audit trail over time units. Next, the syntax trees are obtained from an audit trail, and then the trees are compressed, reduced, and not important ones are eliminated. The discovery stage consists of finding elementary periodic patterns and applying the composition rule to derive the required complex periodic patterns. The computational complexity of search for elementary periodic patterns is approximately $O(k * n^3)$ where $0 < k < 1/8$ and n is the total number of partitions in an audit trail. Complexity of search over syntax trees is hard to estimate as it depends on the total number of access methods to relational tables, complexity of SQL statements, and a level of sharing common components among SQL statements.

The periodicity of database workload provides very useful information for many functional components of a database management system, like transaction scheduler, data buffer cache controller, and automated performance tuner. Usually, such functional components need only partial information about the behavior of selected database applications. An approach in which only some of elementary periodic patterns are discovered and later on composed is more practical as it targets only the specific SQL statements in an audit trail. Another advantage of the proposed approach is the possibility to use the discovered periodic patterns to model future workload after the old applications are replaced with the new ones or the new applications are added to a system. It is also easier to reconcile the new audit trails with the collections of periodic patterns discovered from the previous audit trails than to integrate the complete trails.

References

1. Abadi, M., Manna, Z.: Temporal logic programming. J. Symb. Comput. 8(3), 277–295 (1989)
2. Agrawal, R., Imielinski, T., Swami, A.: Mining association rules between sets of items in large databases. In: Proceedings of the 1993 ACM SIGMOD international conference on management of data, pp. 207–216 (1993)
3. Ahmad, M., Duan, S., Aboulnaga, A., Babu S.: Predicting completion times of batch query workloads using interaction-aware models and simulation. In: Proceedings of EDBT, pp. 449–460 (2011)
4. Akdere M., Çetintemel U., Riondato M., Upfal E., Zdonik B.: Learning-based query performance modeling and prediction. In: Proceedings of ICDE, pp. 390–401 (2012)
5. Baudinet, M., Chomicki, J., Wolper, P.: Temporal deductive databases. In: Tansel, A., Clifford, J., Gadia, S., Jajodia, S., Segev, A., Snodgrass, R (eds) Temporal Databases: Theory, Design, and Implementation. Benjamin/Cummings, pp 294–320 (1993)
6. Bruno, N.: Automated Physical Database Design and Tuning. CRC Press Taylor and Francis Group, Boca Raton (2011)
7. Huang, K.Y., Chang, C.H.: SMCA: a general model for mining asynchronous periodic patterns in temporal databases. IEEE Trans Knowl Data Eng **17**(6), 774–785 (2005)
8. Han, J.W., Gong, W., Yin, Y.W.: Mining segment-wise periodic patterns in time-related databases. In: Proceedings of international conference on knowledge discovery and data mining, pp. 214–218 (1998)
9. Laxman, S., Sastry, P.S.: A survey of temporal data mining. Sadhana Acad Proc Eng Sci **31**(2), 173–198 (2006)
10. Mannila, H., Toivonen, H., Verkamo, A.I.: Discovery of frequent episodes in event sequences. Data Min Knowl Discov **1**, 259–289 (1997)
11. Özden, B., Ramaswamy, S., Silberschatz, A.: Cyclic association rules. In: Proceedings of the 14th international conference on data engineering, pp. 412–421 (1998)
12. Rasheeed, F., Alshalalfa, M., Alhajj, R.: Efficient periodicity mining in time series databases using suffix trees. IEEE Trans Knowl Data Eng **23**(1), 79–94 (2011)
13. Roddick, J.F., Society, I.C., Spiliopoulou, M.: A survey of temporal knowledge discovery paradigms and methods. IEEE Trans Knowl Data Eng **14**(4), 750–767 (2002)
14. Simovici, D.A., Djeraba, C.: Mathematical Tools for Data Mining: Set Theory, Partial Orders, Combinatorics. Advanced Information and Knowledge Processing. Springer, London (2008)
15. Wojciechowski, M.: Discovering frequent episodes in sequences of complex events. In: Proceedings of enlarged 4th East-European conference on advances in databases and information systems (ADBIS-DASFAA), pp. 205–214 (2000)
16. Wu, W., Chi, Y., Hacıgümüş, H., Naughton, J.F.: Towards predicting query execution time for concurrent and dynamic database workloads. Proc VLDB Endow **6**(10), 925–936 (2014)
17. Wu, W., Chi, Y., Zhu, S., Tatemura, J., Hacıgümüş, H., Naughton, J.F.: Predicting query execution time: are optimizer cost models really unusable? In: Proceedings of ICDE, pp. 1081–1092 (2013)
18. Yang, J., Wang, W., Yu, P.S.: Mining asynchronous periodic patterns in time series data. IEEE Trans Knowl Data Eng **15**(3), 613–628 (2003)
19. Zimniak, M., Getta, J.R., Benn, W.: Discovering periodic patterns in database audit trails. In: Proceedings of international conference on interdisciplinary research theory and technology, pp. 365–367 (2013)