Discovering Periodic Patterns in Database Audit Trails

Marcin Zimniak\(^1\), Janusz R. Getta\(^2\), and Wolfgang Benn\(^1\)

\(^1\) Faculty of Computer Science, TU Chemnitz, Germany
{marcin.zimniak,benn}@cs.tu-chemnitz.de
\(^2\) School of Computer Science and Software Engineering,
University of Wollongong, Australia
jrg@uow.edu.au

Abstract. Information about periodic processing of database operations has a pivotal importance for continuous physical database design and automated performance tuning of database systems. This work shows how to detect the oscillations of database workloads caused by the periodical invocations of user applications. In particular, we present an algorithm for discovering the periodic patterns in processing of database operations. In our approach, information collected from the database audit trails is transformed into a sequence of syntax trees and later on it is compressed in a syntax tree table. The periodic patterns are discovered through nested iterations over a four dimensional space of syntax trees and positional parameters of the patterns. Transformations of the patterns are used to discover the overlapping periodic patterns.

Keywords: periodic pattern, database audit trail, automated performance tuning

1 Introduction

Automated performance tuning of database systems [1] requires the prognostics on variations of future database workload as well as the frequencies of access to data containers by the user applications. The periodic iterations of real world processes reflect in a database system as the periodic changes of database workload. Such changes can be discovered from historical information stored in the log files, traces from processing of database applications, audit trails, etc. At the first glance the problem periodic pattern discovery seems to be very similar to the classical problem of periodicity mining in time series [2]. However, due to the internal structures of complex database operations the problem cannot be treated in the same way as analysis of atomic data elements in time series or genetic sequences. The traditional approaches assume 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 that happen in daily, monthly and yearly workload of a database system. Periodically processed user applications have the frequencies consistent with the frequencies of real world events.
The problem of discovering periodic patterns in the database workloads is quite similar, however, it is not exactly the same as a problem of mining cyclic association rules [3]. The objective of mining cyclic association rules is to find the largest sets of operations on data that are periodically processed by a database system. In our case the largest sets of operations do not necessarily mean the highest workload. Additionally, mining of cyclic association rules is not able to discover two or more periodic periodic patterns whose cycles overlap on the same period of time.

To discover the periodic patterns in the processing of elementary and complex database operations we use information about the past "behaviour" of a database system stored in the anonymized audit trails. SQL statements included in audit trails can be processed with EXPLAIN PLAN statement in order to obtain the precise execution plans and estimation of the processing costs. Next, the execution plans can be converted into the syntax trees and saved into a syntax tree table. The table can be further reduced and later on it can be used in the iterations that reveal the periodic patterns in processing of database operations.

The paper is organized in the following way. The next section reviews the major works on periodicity mining in time series and mining cyclic association rules. Section 3 defines an environment of relational database and the concepts of audit trail, syntax tree table, and time units. A concept of periodic patterns in database workload is introduced in Section 4 and discovering periodic patterns is explained in Section 5. Finally, Section 6 concludes the paper.

2 Related work

Information about the processing of theoretically infinite sequence of database applications available from the log files, traces, audit trails, etc., suggest application of data mining techniques based on analysis of ordered set of operations on data performed by the user applications [4]. 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 [5]. Data mining techniques that inspired the works on periodic patterns came from the works on mining frequent episodes [6]. A starting point to many works on discovering cyclic patterns was [3] which used cycle pruning, cycle skipping, cycle elimination heuristics. In the recent years more work on discovering period patterns addressed full periodicity, partial periodicity, perfect and imperfect periodicity [7] and recently asynchronous periodicity [8].

3 Database processing model

In this work, we consider a typical relational database system where a relational model of data is used to represent data containers. Let $x$ be a nonempty set of attribute names later on called as a schema and let $\text{dom}(a)$ denotes a domain of attribute $a \in x$. A tuple $t$ defined over a schema $x$ is a full mapping $t : x \rightarrow \cup_{a \in x} \text{dom}(a)$ and such that $\forall a \in x, t(a) \in \text{dom}(a)$. 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 user applications into the query execution plans formulated as the expressions of extended relational algebra.

### 3.1 Audit trail

A history of SQL processing is stored in a trace from processing of user applications \( a_1, \ldots, a_n \). A trace of a user application \( a_i \) is a finite sequence of pairs \( <c_i: t_{c_i}, s_{i_1}: t_{i_1}, \ldots, s_{i_n}: t_{i_n}, d_i: t_{d_i}> \) where \( c_i \) is a connect statement, \( t_{c_i} \) is a timestamp when the statement has been processed, all \( s_{i_j} \) are SQL statements, all \( t_{i_j} \) are timestamps of the respective SQL statements, \( d_i \) is a disconnect statement, and \( t_{d_i} \) is a timestamp of disconnect statement. An audit trail is a complete trace from processing of many user applications and due to concurrent processing of user applications it is an interleaved sequence of connect, disconnect, and SQL statements.

### 3.2 Syntax tree table

Let \( s_i \) and \( s_j \) be the statements obtained from an audit trail and let \( T_{s_i}, T_{s_j} \) be their respective syntax trees obtained from the applications of \texttt{EXPLAIN PLAN} statement.

We say, that a syntax tree \( T_{s_i} \) is included in or equal to a syntax tree \( T_{s_j} \) and we denote it with \( T_{s_i} \sqsubseteq T_{s_j} \) if there exists a nonleaf 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 \( T_{s_i} \).

Complete information about syntax trees of SQL statements extracted from an audit trail is stored in a syntax tree table. A syntax tree table is a set of tuples \( <\text{tree}, \text{operation}, \text{left}, \text{right}, \text{workload}, \text{timestamps}> \) where \( \text{tree} \) is a unique identifier of a syntax tree, \( \text{operation} \) is a code of extended relational algebra operation at the root of syntax tree identified by \( \text{tree} \), \( \text{left} \) and \( \text{right} \) are the identifiers of left and right argument of syntax tree identified by \( \text{tree} \) or the names of relational tables, \( \text{workload} \) is an estimate workload imposed on a database system when processing a syntax tree \( \text{tree} \), and \( \text{timestamps} \) is a set of all timestamps when a syntax tree \( \text{tree} \) was processed by a database system. 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 where \( p_1, p_2, \) and \( p_3 \) are the codes of operations. The respective syntax tree table is given below.
3.3 Time units

Let $<t_{\text{start}}, t_{\text{end}}>$ 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 $<t^{(i)}_c, \tau_c>$ where $t^{(i)}_c$ for $i = 1, \ldots, n$ is a timestamp when an elementary time unit starts and $\tau_c$ is a length of the unit. Elementary time units are distributed over $<t_{\text{start}}, t_{\text{end}}>$ such that $t_{\text{start}} = t^{(1)}_c$ and $t^{(i+1)}_c = t^{(i)}_c + \tau_c$ and $t^{(n)}_c + \tau_c = t_{\text{end}}$.

A time unit is a pair $<t, \tau>$ where $t$ is a start point of a unit and $\tau$ is a length of the unit. A time unit consists of one or more consecutive elementary time units.

A sequence $U$ of $n$ disjoint time units $<t^{(i)}, \tau^{(i)}>$ $i = 1, \ldots, n$ over $<t_{\text{start}}, t_{\text{end}}>$ 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 $\tau^{(n)} + \tau_{\text{threshold}} \leq t_{\text{end}}$.

4 Periodic patterns

Let $A$ be an audit trail. A periodic pattern is a tuple $<T, U, w, b, e, p>$ where $T$ is a syntax tree, $U$ is a sequence of disjoint time units that partitions the audit trail into disjoint sequences of operations, $w$ is a threshold workload, $b$ is a number of time unit in $U$ where the pattern begins, $e$ is a number of time unit in $U$ where the pattern ends, and $p$ is a period of pattern measured as the total number of time units+1 between any two successive computations of $T$. Let $|U|$ denotes the total number of time units in $U$. Then, the positional parameters $b, e, p$ in a periodic pattern must satisfy the following properties: $1 \leq b < e \leq |U|$ and $\exists n \in 1, 2, \ldots, e = b + n * p$ and threshhold workload $w > 0$.

A sequence $W_T$ is called as workload histogram of a syntax tree $T$ and it is used to represents the workloads imposed on a database system when processing a syntax tree $T$ in each time unit in $U$. The $n$-th value in a workload histogram $W_T[n]$ is equal to $w_T * |T\text{.timestamps}[n]|$, where $T\text{.timestamps}[n]$ is a set of timestamps included in the $n$-th time unit and associated with the identifier of a syntax tree $T$ in a syntax tree table.

Consider a periodic pattern $<T, U, w, b, e, p>$. Let $U_T$ be a sequence of time units with numbers $b, b+p, b+2p, \ldots, e$ in $U$. Let $U_T^\perp$ be a sequence of time units in $U_T$ and such that total workload created by processing of $T$ in each
unit in $U^+_T$ is greater than $w$. In the other words, $U^+_T$ consists of time units in $U_T$ with numbers $n$ such $W_T[n] \geq w$. Then, we say that a periodic pattern $<T, U, w, b, e, p>$ is valid in an audit trail $A$ with a support $0 < \sigma \leq 1$ if $W[b] \geq w$ and $W[e] \geq w$ and $\sigma \leq |U^+_T|/|U_T|$. 

5 Discovering periodic patterns

Discovering periodic patterns can be performed 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 $w$, and the dimensions of positional parameters $b$, $c$, and $p$. In this work we assume the given values of $U$ and $w$ and we search for periodic patterns over the dimensions of syntax trees and positional parameters.

5.1 Reduced syntax tree table

Let $T$ be a set of syntax trees that consists of all syntax trees of statements in an audit trail. Let $T_e$ be an empty syntax tree and let $T_\pi$ 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 periodic patterns in an audit trail is performed over a lattice $<T, \sqsubseteq>$ implemented as a syntax tree table with a minimum $T_e$ and maximum $T_\pi$. The following three rules can be used to reduce the total number of iterations over the syntax trees.

1. If a periodic pattern $<T_s, U, w, b, e, p>$ occurs in an audit trail $A$ then for any syntax tree $T$ such that $T \sqsubseteq T_s$ the same periodic pattern occurs in $A$.
2. If a periodic patterns $<T_s, U, w, b, e, p>$ does not occur in an audit trail $A$ then for any syntax tree $T$ such that $T_s \sqsubseteq T$ the same pattern does not occur in $A$.
3. If a periodic patterns $<T_s, U, w, b, e, p>$ does not occur in an audit trail $A$ then for any syntax tree $T \sqsubseteq T_s$ and not shared with any other subtree the same pattern does not occur in $A$.

The rules listed above mean that for any syntax tree $T \sqsubseteq T_s$ and not shared with any other subtree a set of periodic pattern that occurs in $T$ is the same as set of periodic patterns that occur in $T_s$. It allows to reduce a syntax tree table to a simple table of pairs $<\text{tree}, \text{timestamps}>$ where $\text{tree}$ is an identifier of a syntax tree that suppose to be verified against periodic patterns and $\text{timestamps}$ is a set of timestamps when the processing of a syntax tree identified by $\text{tree}$ occurred in an audit trail. For example, a syntax tree table given in Table 1 reduces to a set of pairs $\{<1, \{ts_1, ts_3, ts_4\}>,, <2, \{ts_1\}>,, <3, \{ts_2, ts_4\}>,, <5, \{ts_4\}>\}$.

5.2 Iterations

Discovering periodic pattern $<T, U, w, b, e, p>$ for a given set of time units $U$, a given minimal workload $w$, 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 \( b \), \( e \), and \( p \). At the beginning all syntax trees in a reduced syntax tree table are marked as "not processed yet" and a set \( \mathcal{P} \) of periodic patterns that occur in an audit trial \( A \) is set to empty. At each level the iterations are performed in the following way.

(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 histogram \( W_T \) for \( T \).

(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 \( 1, 2, 3, \ldots, |U| - 1 \). Let \( b_c \) be the current value of parameter \( b \). If \( W_T[b_c] \leq w \) 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.

(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 \( |U|, |U| - 1, \ldots, b_c + 2, b_c + 1 \). Let \( e_c \) be the current value of parameter \( e \). If \( W_T[e_c] \leq w \) 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 level (1.1).

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

(1.1.1.2) Next, we create a candidate periodic pattern \( <T, U, w, b_c, e_c, p_c> \) and we use a histogram \( W_T \) to check whether the candidate pattern is valid in an audit trail with a given support \( \sigma \). Let \( W_T^+ \) be a set of all values of histogram \( W_T \) with the same numbers as a set of time units \( U_T \) and such each value in \( W_T^+ \geq w \). Then, a candidate periodic pattern \( <T, U, w, b_c, e_c, p_c> \) is valid in an audit trail with a support \( \sigma \) when \( W[b_c] \geq w \) and \( W[e_c] \geq w \) and \( \sigma \leq |W_T^+|/|U_T| \).

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 \( p \).

(1.1.1.3) If a candidate pattern is valid in an audit trail then we find the smallest workload in the pattern \( w_{\min} = \min\{W_T[b_c + i * p_c], \forall i = 0, 1, 2, \ldots, (e_c - b_c)/p_c\} \) and we append \( <T, U, w_{\min}, b_c, e_c, p_c> \) to a set \( \mathcal{P} \). Then we modify the terms of histogram \( W_T \) such that \( W_T[b_c + i * p_c] := W_T[b_c + i * p_c] - w_{\min}, \forall i = 0, 1, 2, \ldots, (e_c - b_c)/p_c \). Next we return to step (1.1.1.1) to collect the next value of parameter \( p \).

(1.2) At the end of iterations over the positional parameters we are left with the single elements in a workload histogram \( W_T \), which are not attached to any periodic pattern in \( \mathcal{P} \) and such that their value is greater than \( w \). If there exists a periodic pattern \( <T, U, w, b, e, p > \in \mathcal{P} \) and an element \( W_T[n] > w \)
such that \( n \in \{ b + p, b + 2 \times p, \ldots, e - 2 \times p, e - p \} \) then we split the pattern into \( < T, U, w, b, n, p > \) and \( < T, U, w, n + p, e, p > \) and we modify histogram 
\( W_T[n] := W_T[n] - w \). Splitting of 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.

6 Conclusions and further work

The efficiency of search over the dimensions of syntax trees and positional parameters of periodic pattern is very low. If an audit trail is divided by a set of time units \( U \) into \( n \) partitions then complexity of a search over the values of \( b, e, p \) is approximately \( O(k \times n^3) \) where \( 0 < k < 1/8 \). 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.

As usual more efficient search over a space of syntax trees and positional parameters is a natural objective for the further research. As an ultimate objective is to apply the discovered periodic patterns to automated database performance tuning the next open problem is an application of the patterns to the prognostics of future intensity and structure of database workload. It requires a system of derivation rules for the periodic patterns to estimate what relational tables will be accessed by user applications and what database operations will be processed by the applications. One more issue is the right choice of a sequence of time units \( U \) when searching for periodic patterns. Too long time units in \( U \) would hide the existence of periodic patterns while too short time units would make the discovered patterns hard to comprehend and not consistent with the reality.

References