Advances in Computational Sciences and Technology
ISSN 0973-6107 Volume 10, Number 5 (2017) pp. 945-963
© Research India Publications
http://www.ripublication.com

Veiled Endpoints: A Novel Dimension for Aggregation

Shailender Kumar[1], Nitish Sharma[2] and Rupender Duggal[3]
[1,2,3] Department of Computer Science and Engineering,
Ambedkar Institute of Advanced Communication Technologies & Research, Delhi, India

Abstract

In the coeval contexture a hefty peck of the prevalent database employments concert with the temporal data. Exegesis of this fain gesture for the temporal data comes from the temporal aspects as they provision peculiar traits and tasks for query processing and optimization. The substantial challenge in these tasks lies in computing the temporal aggregates which itself is an intricate aspect considering the grouping of temporal extents. Temporal grouping environs the partitioning of time line and for each partition aggregate functions are applied separately. This paper tenders an unprecedented blueprint to aggregate temporal extents by recognizing the domains that are implicitly veiled. These veiled regions turn out to be very efficacious while applying the explicitly defined aggregate functions which depend on the extent of coupling amongst the dynamic attributes. The paramount purpose of applying aggregate functions along with B-Tree indexing is achieved by significant improvement in the query execution time and number of logical reads.

Keywords: Aggregates, Data model, Databases, Query processing, Run time

INTRODUCTION

Database employments frequently need to record the variation in behavior of the time oriented attributes. The significance of temporal aspects has been acknowledged by numerous research groups. As a consequence various query languages which provisions temporal aspects of an object have been developed so far. The substantial challenge in query processing and optimization lies on the computation of temporal aggregates which itself is an intricate aspect considering the grouping of temporal extents. Temporal grouping involves the segmentation of a time-line and the tuples corresponding to the particular time segments are grouped together [1]. Then, temporal aggregate functions are applied identically on each time segment to obtain the aggregate result.

Temporal aggregation is theoretically more complex as compared to duplicate elimination, which relies on an equality predicate applied over the attributes. It involves detection of the overlapping tuples, which is an inequality predicate over the timestamp attribute. Temporal aggregation plays a major role in optimization of numerous temporal data models which encloses tuple time-stamped as well as attribute time-stamped data models [2,3]. Value equivalent tuples are merged as a result of aggregation operation. Aggregation is an essential operation in temporal databases since evaluation of temporal queries on un-coalesced temporal entities can produce inappropriate results. Aggregate operations are irreversible in nature for it is difficult to reconstruct the primary set of intervals or values from the aggregated result set.

Usually, it is difficult to switch between a coalesced and an un-coalesced representation of a relation without altering the semantics of a procedure [4]. Temporal grouping or aggregation of tuples is broadly divided into two flavors: span temporal aggregation and instant temporal aggregation. In instant temporal aggregation, the aggregated result set at a time instant ‘t’ is obtained by computing over the set of all those tuples whose timestamp includes t. On the other hand, Span temporal aggregation is integrally centered on explicitly defined time interval, such as hours, days, weeks or months and doesn’t depend on the value of temporal attributes of a relation.

The computation of temporal aggregates can be achieved through these two manners: either sequential or parallel. The parallel computation techniques are frequently employed for the scope and size of data-intensive employments have matured significantly as evinced in online analytical processing (OLAP) and data warehousing environments [1,5]. Although numerous sequential as well as parallel procedures have been developed for computing temporal aggregates, yet they need prior information about the order or size of an input.

In this paper, we introduce a novel technique for temporal aggregation of value equivalent tuples by using explicitly defined aggregate functions along with B-Tree indexing to minimize the query execution time. This significant improvement in query execution time affects the performance of underlying temporal data model to a great extent. Query execution time is optimized by further indexing of the non key attributes. Overlapping, adjacent or nearby tuples are the candidates for temporal aggregation. Main emphasis of our work is to perform aggregation of those tuples which are separated by temporal gaps. Aggregation of value equivalent tuples depends on the nature of time-varying attributes as well as on the extent of coupling amongst the time-varying attributes. Our research work uses the most powerful open source database postgres for the performance evaluation of the proposed approach.

The rest of this paper is organized as follows. Next section surveys the background and related work on computing temporal aggregates. Major limitations of previous work are also discussed in the section. In Sections 3, and 4, we present the improved algorithm for aggregation and scalable solutions for large-scale aggregations based on data partitioning and parallel processing techniques. Section 5 presents the results of experimental evaluation of the proposed solutions. Finally, Section 6 summarizes the contributions of this paper and gives an outlook to future work.

RELATED WORK

Aggregation of value equivalent tuples can be done by using either scalar aggregate functions or aggregate procedures. Scalar aggregate function such as min, max, count and sum yields a single tuple over an entire relation [3,5]. On the other hand, aggregate procedures involve partitioning of a complete relation on the basis of some specific attributes of a temporal relation. Then, computation of scalar aggregate functions is carried out individually over each of the partition of a relation. It is difficult to manage more than one scalar aggregate expression in a query. Aggregated result of every expression is stored in a singleton relation which is used as a base relation for the computation of rest part of the temporal query [6].

Transformation procedures are employed for the generation of systematically efficient algorithms intentionally designed for the computation of temporal aggregate queries. A strategy for the aggregation of value equivalent tuples involves two steps for the aggregation over the entire temporal relation which entails perusing of a database. Aggregate is evaluated over the constant intervals which are determined in the initial step.

Numerous strategies for the aggregation of tuples have been developed so far [7,8]. They basically differ on the basis of approach used for the partitioning of a time line. Instant temporal aggregation maneuvers at the smallest granularity of a time instant. Undistinguishable aggregate outcomes at successive time instants are aggregated into single tuple over maximal time intervals. Moving window temporal aggregation is basically based upon instant temporal aggregation. It extends the scope of aggregation from time instant to window of time interval. The major shortcoming of both the aggregation technique is that the result size might become up to twice as large as the input size. Span temporal aggregation allows the user to regulate the result size of aggregate operation by partitioning the time line into several time intervals that are specified in the query [10,11,12].

A uniform framework was developed to standardize aggregate operation in temporal environment. It supports the empirical evaluation of several aggregation techniques or temporal aggregation operators. R-tree is used to evaluate the aggregate operation in linear space and logarithmic time with respect to the size of the database [3,5,12]. Choice of underlying temporal data model significantly impacts the performance of the aggregate procedures. Parsimonious temporal aggregate operator was introduced to overcome the shortcomings of instant temporal aggregate operator [8]. It takes the result of instant temporal aggregation of size n and allows the user to regulate the trade-off between the result size and error introduced by aggregation of tuples.

PRELIMINARIES

A temporal relation contains finite set of attributes along with finite set of domains and timestamps. Temporal procedures are used to build a relationship between the time varying attributes and their particular domains [3,5].

Temporal relational schema

A collection of key attributes along with time variant and time invariant attributes are present in any temporal relation. Temporal relational schema has been demonstrated mathematically in the equation (1) as given below:

$$R^t = (K_1, K_2, \dots, K_n, S_1, S_2, \dots, S_n, D_1, D_2, \dots, D_n | T) \quad (1)$$

Temporal Aggregation Approaches

Aggregation of value equivalent tuples is carried out through these three approaches as shown in Table 1. It primarily depends on the system environment which includes underlying temporal data model, domain of time varying attributes and extent of coupling amongst the time varying attributes.

Table 1: Aggregation approaches

Event	**Approach**
Run-time	During execution of query, tuples are aggregated as required
Insert	During insertion of tuple in the relation, tuples are aggregated
Update	Tuples are aggregated, when existing data is modified

Temporal Relational Algebra

Temporal relational algebraic expression includes temporal operators and predicate symbols. These expressions are used to query temporal databases for the retrieval of information. Temporal operators differs from the traditional relational operators as they provide implicit support for the handling of time dimension in temporal data model. It is mandatory to use temporal relational algebra in such a way that it does not violate any integrity constraints.

Temporal Predicates

Classification of temporal predicate are: (1) Interval predicates, (2) Join predicates and (3) element predicates. Join predicates and element predicates are nothing but the generalization of the first category of predicates called as interval predicates. It is appropriate to use element predicate for aggregated relations as its complexity is also high as compared to join predicates [5,6].

Temporal Constructors

Temporal constructors are used to project time from a set of tuples. It is frequently used for the partial aggregation of operands. Some basic types of temporal constructors are: Projection, Intersection, Begin and End.

Definition 1

(Aggregation operator) Let r be a temporal relation with schema $R = (G_1, G_2, G_3, \dots, G_n, V_1, V_2, V_3, \dots, V_m \mid T)$, where $G = \{G_1, G_2, G_3, \dots, G_n\}$is the set of time invariant attributes and $V = \{V_1, V_2, V_3, \dots, V_m\}$ is the set of time variant attributes. The aggregation of two adjacent tuples, $r_i, r_j \in r,\ r_i < r_j$ is given by:

$$r_i \oplus r_i = (r_i.G_1, r_i.G_2, r_i.G_3, \dots, r_i.G_n, r_j.V_1, r_j.V_2, r_j.V_3, \dots, r_j.V_m | r_i.t_s, r_j.t_e) \quad (2)$$

Definition 2

(Overlapping tuples) Let r be a temporal relation with schema$R = (G_1, G_2, G_3, \dots, G_n, V_1, V_2, V_3, \dots, V_m \mid T)$. Two tuples, $r_i, r_j \in r$are overlapping, iff the following holds true:

$(1) r_i.V = r_j.V$

$(2) r_i.t_e > r_j.t_s$

The first expression requires that the tuples have similar time invariant attributes. While the second expression demands that the validity end time of first tuple is greater than the validity start time of second tuple.

Definition 3

(Adjacent tuples) Let r be a temporal relation with schema $R = (G_1, G_2, G_3, \dots, G_n, V_1, V_2, V_3, \dots, V_m \mid T)$. Two tuples, $r_i, r_j \in r$are adjacent, iff the following holds true:

$(1) r_i.V = r_j.V$

$(2) r_i.t_e = r_j.t_s$

The first expression compels that the tuples must have similar time invariant attributes, whereas the second expression requires that the validity end time of first tuple should be equal to the validity start time of second tuple.

Definition 4

(Nearby tuples) Let r be a temporal relation with schema$R = (G_1,\ G_2\ ,\ G_3, \dots, G_n, V_1,\ V_2\ ,\ V_3, \dots, V_m \mid T)$. Two tuples, $r_i, r_j \in r$are nearby, iff the following holds true:

(1)$r_i.V = r_j.V$

(2)$r_i.t_e < r_j.t_s$

The first expression requires that the tuples have similar time invariant attributes. The second expression puts a constraint that the validity end time of first tuple should be smaller than the validity start time of second tuple. Here, we need to fix the value of temporal gap between any two value equivalent tuples. The temporal gap is defined as the time gap between any two tuples of a relation. Temporal gap is calculated by using equation (3).

$$\text{Temporal gap}(\alpha) = r_j.t_s - r_i.t_e \qquad (3)$$

The time difference between two timestamps is calculated by using implicit temporal functions which are used for the handling of temporal attributes. Here, value of temporal gap (α) defines the possibility of aggregation.

AGGREGATION

In this segment, aggregation operation is introduced. Every tuple of a temporal relation doesn’t act as a candidate tuple for aggregate operation. Tuples recognized as a suitable candidate for aggregate operation are captured from the relation. Then, the temporal grouping of captured tuples is performed on the basis of start or end time. At last, aggregate procedures are evaluated over temporal groups present in a relation and the corresponding results are stored in a singleton relation [7,8]. The two extremes in the spectrum of approaches for aggregation are runtime aggregation and insert or update aggregation. The primary factor which affect our decision to select one of the above said approaches is system environment which includes underlying temporal data model, extent of coupling amongst the time varying attributes, type of time-stamping used and domain of time varying attributes.

In our proposed work, update or insert strategy is employed for the aggregation of value equivalent tuples. Explicitly defined aggregate procedures are developed to perform aggregate operation. During update, aggregated result set is captured from the base relation and aggregate procedures are evaluated over the captured snapshot of the base relation. It involves modification of time varying attributes as well as shortening or extension of time line. Aggregate procedures defined for aggregation of tuples are capable enough to perform aggregation of all the overlapping, adjacent and nearby tuples. These procedures also ensures that the basic properties of aggregation like extent preservation, maximal interval or duplicate elimination are preserved during aggregation of tuples. Veiled regions are also identified during execution of these temporal aggregation procedures [5,9]. Temporal predicates and constructors are used to project time from a set of tuples and also used for partial aggregation of operands.

Due to heterogeneous nature of time varying attributes, distinct aggregate procedures are employed for every time varying attribute. Aggregated result set of each of the dynamic attribute is stored separately in aggregate relations. Number of aggregate relations formed are equal to the number of time varying attributes present in the temporal relation schema.

Properties

The aggregation operation should encompass the following three properties: (1) Extent preservation, (2) Maximal intervals and (3) Duplicate elimination. Extent preservation ensures that the temporal extent of a tuple should not change after the execution of temporal aggregation operators. Maximal interval enforce that aggregation should produce output tuple with maximal time periods. And, duplicate elimination impose a constraint for the removal of duplicate tuples from the snapshot relation [3,5].

Extent preservation: Aggregation must not alter the temporal extent of a timestamp.Denote the extent of S as: $\mathrm{extent(S)} = \{\mathrm{q} | \exists \mathrm{b} \exists \mathrm{e}([\mathrm{s}, \mathrm{e}] \in \mathrm{S} \wedge \mathrm{s} \leq \mathrm{q} \leq \mathrm{e})\}$

and, $\mathrm{extent(S)} = \mathrm{extent}\big(\mathrm{aggregate(S)}\big)$.

Maximal intervals: Aggregation should produce a tuple with intervals that are as large as possible.

Let S and X be timestamps.

Such that, $\mathrm{extent(S)} \subseteq \mathrm{extent(X)}$.

Role of Relational Algebra in Aggregation

Relational algebra enlightens the significance of recognition of veiled endpoints in aggregate operation. It involves aggregation of value equivalent tuples by using single temporal dimension along with preserving the essential properties of aggregation. Steps involved in evaluation of aggregate operation using relational algebra are: (1) Identification of covered endpoints, (2) Identification of uncovered endpoints, (3) Construction of candidate intervals, (4) Candidate identification and (5) Removal of overlapping candidates.

Strategic flow

A strategic flow of our approach for aggregation of value equivalent tuples is illustrated in Fig. 1. It shows the interaction between the temporal relations and the temporal procedures.

The temporal procedures are designed to perform aggregation of tuples and their retrieval from the base relations. Removal of malign or duplicate tuples from snapshot of projected relation involves various steps like identification of covered endpoints, temporal grouping, construction of candidate intervals,

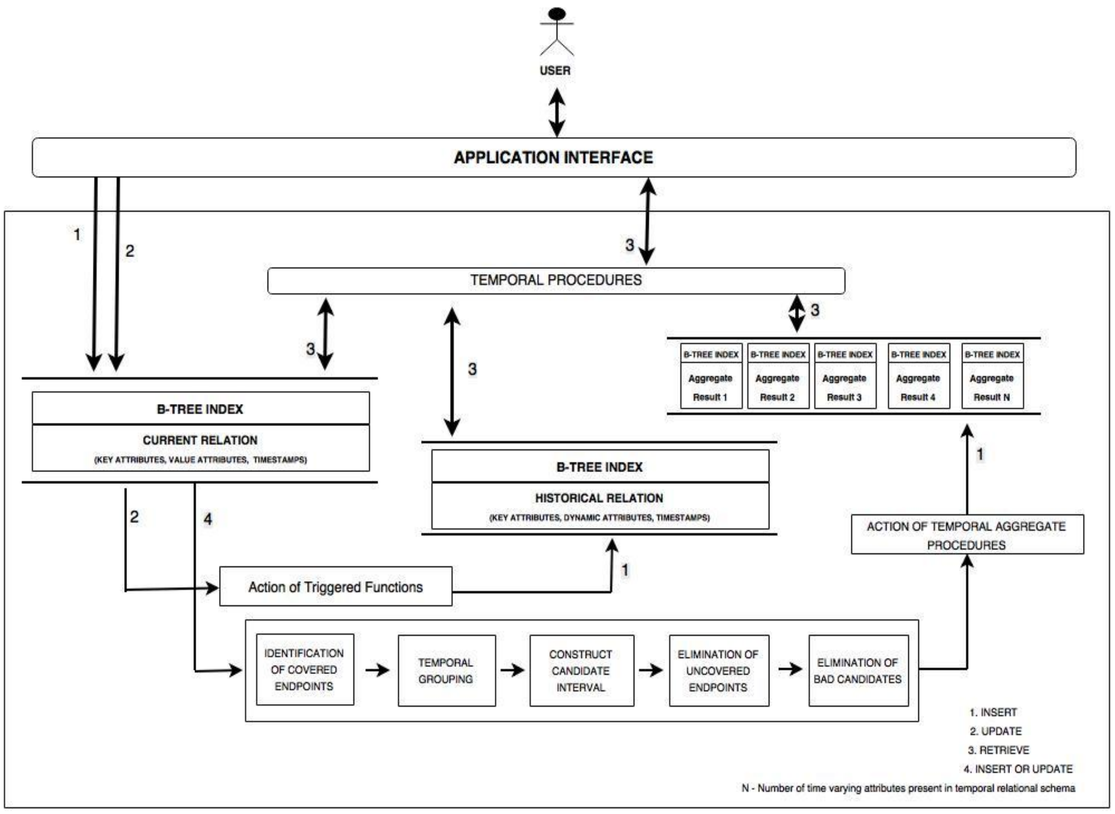

USER
APPLICATION INTERFACE
1
2
3
TEMPORAL PROCEDURES
3
3
3
B-TREE INDEX
Aggregate Result 1
B-TREE INDEX
Aggregate Result 2
B-TREE INDEX
Aggregate Result 3
B-TREE INDEX
Aggregate Result 4
B-TREE INDEX
Aggregate Result N
B-TREE INDEX
CURRENT RELATION
(KEY ATTRIBUTES, VALUE ATTRIBUTES, TIMESTAMPS)
B-TREE INDEX
HISTORICAL RELATION
(KEY ATTRIBUTES, DYNAMIC ATTRIBUTES, TIMESTAMPS)
1
2
4
ACTION OF TEMPORAL AGGREGATE PROCEDURES
Action of Triggered Functions
1
IDENTIFICATION OF COVERED ENDPOINTS
TEMPORAL GROUPING
CONSTRUCT CANDIDATE INTERVAL
ELIMINATION OF UNCOVERED ENDPOINTS
ELIMINATION OF BAD CANDIDATES
1. INSERT
2. UPDATE
3. RETRIEVE
4. INSERT OR UPDATE
N - Number of time varying attributes present in temporal relational schema

Figure 1: Strategic flow

elimination of uncovered endpoints and, elimination of malign candidates. A dedicated temporal aggregate procedure is designed for each time varying attribute and they implicitly controls the flow of data between the temporal relations. Aggregate procedures are activated when insert or update operation is triggered over current relation of any time varying attribute. Implicit temporal operators are used to perform time oriented calculations. Validity of temporal extents is also checked, to enforce integrity constraints. Several comparisons of timestamps over multiple temporal relations are performed [10,11]. Aggregate procedures are applied along with B-Tree indexing to improve query execution time. It will impact the overall performance of the underlying temporal data model.

Identification of malign candidates

Consider a temporal relational schema T(k, v, s, e) where k represents a set of key attributes, v represents a set of value attributes, s is the validity start time, and e is the validity end time. Aggregation generates a temporal relation schema with similar set of attributes. A veiled point is a point that is contained in the interval of value equivalent tuple in T. Firstly, we need to identify veiled regions present in our temporal relation. Note that T′ is nothing but a renamed version of T.

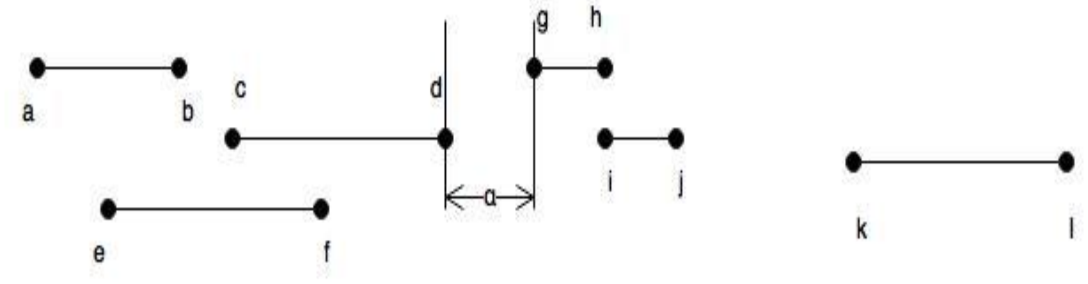


Figure 2: Intervals to aggregate

$$\mathrm{Veiled_s} = \pi_{\mathrm{T.v},\ \mathrm{T.s}}(\sigma_{(\mathrm{T'}.s < T.s \le \mathrm{T'}.e)}(\mathrm{T} \bowtie_{\mathrm{T.v=T'.v}} \mathrm{T'}))$$
$$\mathrm{Veiled_e} = \pi_{\mathrm{T.v},\ \mathrm{T.e}}(\sigma_{(\mathrm{T'}.s < T.e \le \mathrm{T'}.e)}(\mathrm{T} \bowtie_{\mathrm{T.v=T'.v}} \mathrm{T'}))$$

It is pretty valuable to recognize the veiled endpoints. They are identified by simply differentiating the veiled region from projected snapshot of the temporal relation.

$$\mathrm{ucov_s} = \mathrm{DIFFERENCE}(\pi_{\mathrm{Tv.s}}(\mathrm{T}), \mathrm{Veiled_s})$$
$$\mathrm{ucov_e} = \mathrm{DIFFERENCE}(\pi_{\mathrm{Tv.e}}(\mathrm{T}), \mathrm{Veiled_e})$$

The next step is to construct candidate intervals from the set of uncovered endpoints. Candidate interval plays a major role in the removal of duplicate tuples from the temporal relation.

$$\text{candidate (cand)} = \mathrm{ucov_s} \bowtie \mathrm{ucov_e}$$

The fourth step is to identify candidates that extend another candidate and it involves join operation as well as intersection between uncovered regions.

$$\text{Bad candidates}(\mathrm{B_s}) = \text{cand} \ltimes_{(\mathrm{cand.v=ucov_s.v \wedge cand.s<ucov_s.s \wedge ucov_s.s \le cand.e})} \mathrm{uncov_s}$$
$$\text{Bad candidates}(\mathrm{B_e}) = \text{cand} \ltimes_{(\mathrm{cand.v=ucov_e.v \wedge cand.s<ucov_e.e \wedge ucov_e.e \le cand.e})} \mathrm{uncov_e}$$

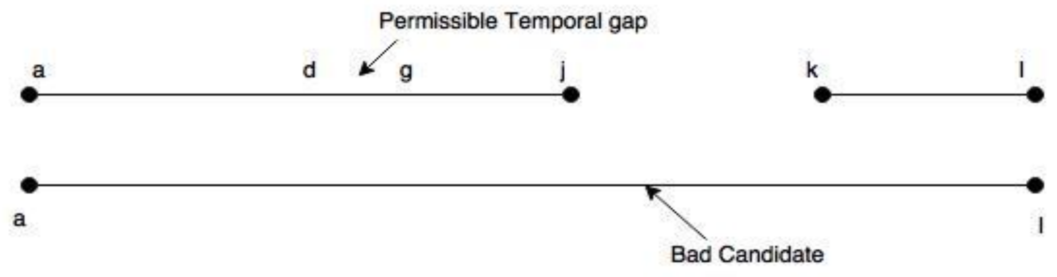


Figure 3: Elimination of bad candidates

The next step is to eliminate the malign, overlapping candidates from the other candidates, leaving the aggregated intervals.

$$\mathrm{Aggregate(T) = DIFFERENCE(cand, Union(B_s, B_e))}$$

Two intervals, [a, j] and [k, l] shown in Fig. 3 result from aggregate operation over the intervals shown in Fig. 2. The veiled relation may cover numerous tuples linked to the similar endpoints since an endpoint can be covered in many ways. Finally, aggregated result set is obtained by simply differentiating the candidate interval from union of malign candidates over start or end time [12]. Aggregated result doesn't contain any duplicate tuples, malign or overlapping candidates. The final result set is stored in auxiliary relations which are associated with a base relation [13].

Maintaining the aggregate relations

The tuples present in aggregate relations must be updated every time when there is an insertion or modification of tuples in a base relation.
The overall size of the single aggregate relation is $O(n * k^2)$
Where,
'n' represents the number of tuples present in the base relation
'k' represents the size of a tuple-group

Insertion into the base relation will add tuples to the aggregate relation. The number of tuples inserted is directly proportional to the overall size of the tuple-group for the inserted tuple [14,15]. Modification of a tuple in the base relation will trigger changes to the aggregated tuples present in aggregate relation.

Pseudo aggregate function

Assume that $\mathrm{R(k, v, s, e)}$ and $\mathrm{R'(k, v, s, e)}$ are two temporal relational schema

Where,

$\mathrm{Key\ attributes(k) = \{k_1, k_2, k_3, \dots, k_m\}}$

$\text{Value attributes}(v) = \{v_1, v_2, v_3, \dots, v_n\}$

's' represents start time(lower bound) of an interval

'e' represents end time(upper bound) of an interval

Pseudo aggregate function for the aggregation of overlapping, adjacent or nearby value equivalent tuples.

```
Function aggregate(R)
$R := R \text{ order by } v_1, v_2, v_3, \dots, v_n, s$
$R' = \text{empty}$
$\text{size} = \text{count}(R)$
$i = 1$
repeat
$R_i = (v_1, v_2, v_3, \dots, v_n)[s, e]$
while
$R_{i+1} = ((v_1 = v'_1, \dots, v_n = v'_n) \quad \text{AND} \quad (s' \leq e + 1 \quad \text{OR} \quad \text{TDiff}(s', e) \leq \alpha))$
$e := \max(e, e')$
$i := i + 1$
while $i \leq \text{size}$
return $R'$
```

Algorithm used for aggregation

Step 1: Define temporal aggregate relation Aggre(key, Tvary_attr, S_Time, E_Time) to capture the snapshot of aggregated result set, initially empty

Step 2: For every entry in temporal relation

Step 3: if T.key not equals to any Aggre.key

Step 4: do

Step 5: insert tuple T into aggregate relation

Step 6: else

Step 7: //Update operation is executed

Step 8: //During update operation: temporal aggregated procedures are activated

Step 9: do

Step 10: Recognition of veiled regions

Step 11: then

```
Step 12: Temporal grouping of recognized tuples are perform
Step 13: // New set of Tuple T
Step 14: if T.S_Time < Aggre.E_Time OR T.S_Time= Aggre.E_Time OR TDiff(Aggre.E_Time,T.S_Time) <= α
Step 15:then
Step 16: Update Aggre.Tvary_attr with T.Tvary_attr
Step 17: Also, Update Aggre.E_time with T.E_time
Step 18: else
Step 19: New tuple is inserted into aggregated relation
Step 20: end if
Step 21: end if
Step 22: end for
Step 23: Output the tuple in Aggre relation.
```

Firstly, we need to recognize veiled endpoints simply by projecting the snapshot of temporal relation schema. Temporal operators are used to calculate the temporal gaps between the value equivalent tuples [16,17]. If the temporal gap between any two value equivalent tuple is less than or equal to the user defined permissible limit then aggregate operation is performed on these tuples. Otherwise, a new set of tuples are inserted into the aggregated relation with new set of value attributes [18]. After that, shortening or extension operation is performed on time line on the basis of end time of new set of tuples.

PERFORMANCE EVALUATION

The proposed approach for aggregation of value equivalent tuples primarily includes recognition of veiled endpoints. A set of temporal operators are employed to provide the basic temporal aspects to the underlying temporal data model. Uncovered endpoints as well as malign candidates are eliminated from the snapshot of the projected temporal relation [19,20]. Time-stamping of the records of the database has been appended by using the time range data type ‘tsrange’ in postgres.

Experimental settings

The experiments are accomplished on the machine which comprises of qualities like 2GB of RAM and Intel(R) Core(TM) i3-3217U 1.8GHz. PostgreSQL of version 9.5 which is an open source database has been thoroughly used while implementing our described approach. The machine was kept in the state of isolation while empirical evaluation of above described strategies. The empirical evaluation is performed during the testing phase where only underlying database management system, a single

SQL shell, and standard circumstantial processes werein live operation. The parameters which are otherwise considered as an out-of-box has been used here as standard parameters for all the database objects. This aspect of our work makes it unique in itself. “pgAdmin” which is a tool of postgres has been used for gathering the data related to query execution time. In our experiments elapsed time which is a significant parameter has been measured after considering the disk access time associated. To achieve accuracy in our measurements this elapsed time of many threads has been aggregated besides neglecting those threads which are of extreme nature [21,22]. To be on the safer side, we neglected the machine cache effects for disk accesses by supplying immaterial data into the complete memory between successive executions of experiments.

Data set

To evaluate our approach on the basis of performance, a huge data set of over one lakh employees has been used to perform the testing of queries. Redundancy has been a serious challenge since long time for the database researchers and in our approach we have tried to exterminate it by decomposing the temporal relations. Here, decomposition is solely based on the number of time varying attributes present in the base relation. If we discuss about the life spans of the tuples used in our approach they are basically of two types named as short-lived and long-lived. The life span of a short-lived tuple lies in a random range of one to 1,000 instants, while the life span of a long-lived tuple is determined approximately between 20,000 and 30,000 instants. Additionally, the start time of these tuples are homogenously dispersed over the entire database time-line. The timestamps used are not of sorted nature through any angle in our experiments.

Parameters involved

We examined all of the above stated approaches made it sure that they stand true even when the conditions varied. We considered three parameters namely relation size, tuple group size, and time dimension size which significantly impact the performance of the underlying temporal data model [23,24]. They also affect the cost of the aggregation operation associated with tuples. If any of these parameters are increased then they linearly affect the number of veiled endpoints and the cost of aggregation [25,26]. For example, if the intensity of overlaps among the intervals increases then it directly impacts the performance as they lead to increased veiled points and coalesced timestamps.

Results

Experiments are carried out for real time testing offollowing representative operations: overlaps, adjacent and nearby. These three operations serve as boundary predicates as they are comparativelymeek to assess for all the aggregation methods.

Usefulness of these operations can be assessed from the fact that they single handedly provide us the comparison of a particular interval against each of the timestamp present in the data table.

A performance evaluation of proposed approachwith respect to runtime approach is performed over tuple timestamp historical relation and tuple timestamp multiple historical relationdata model. Experiments are performed over multiple or large sized value columns. Our goal is to test both the approaches under different conditions. The number of key and value attributes are fixed in temporal relation schema for testing over these two temporal relation data model. Performance of aggregation approaches is evaluated on the basis two parameters: query execution time and number of logical reads.

The schema of every temporal relation is given below:

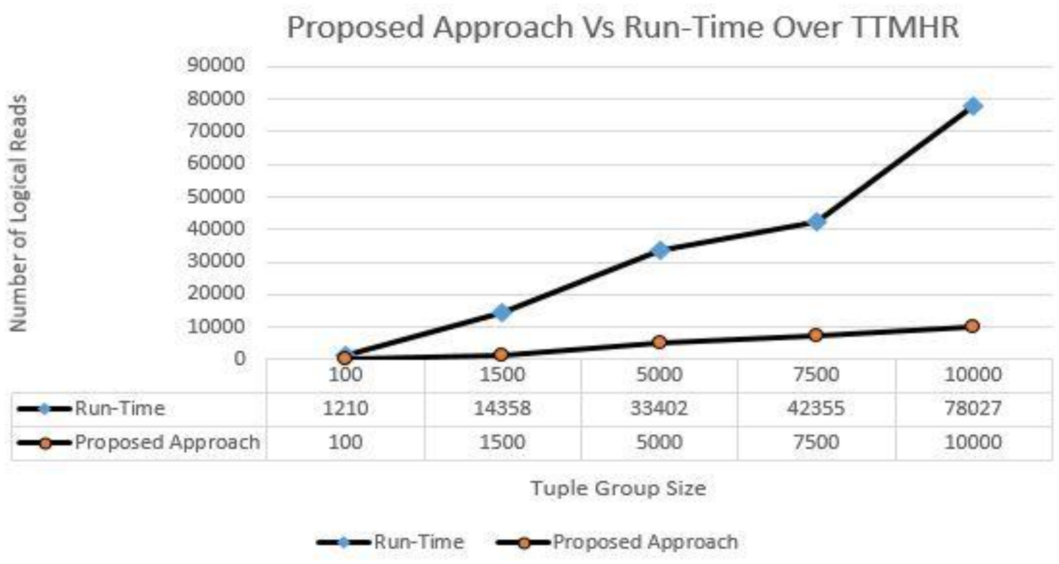
Key - A unique identifier is assigned to every tuple.

Value attributes – It includes time variant or invariant attributes

Start time - The start point of the time interval and it is represented by using lower part of a time range.

End time - The end point of the time interval and it is represented by using upper part of a time range.

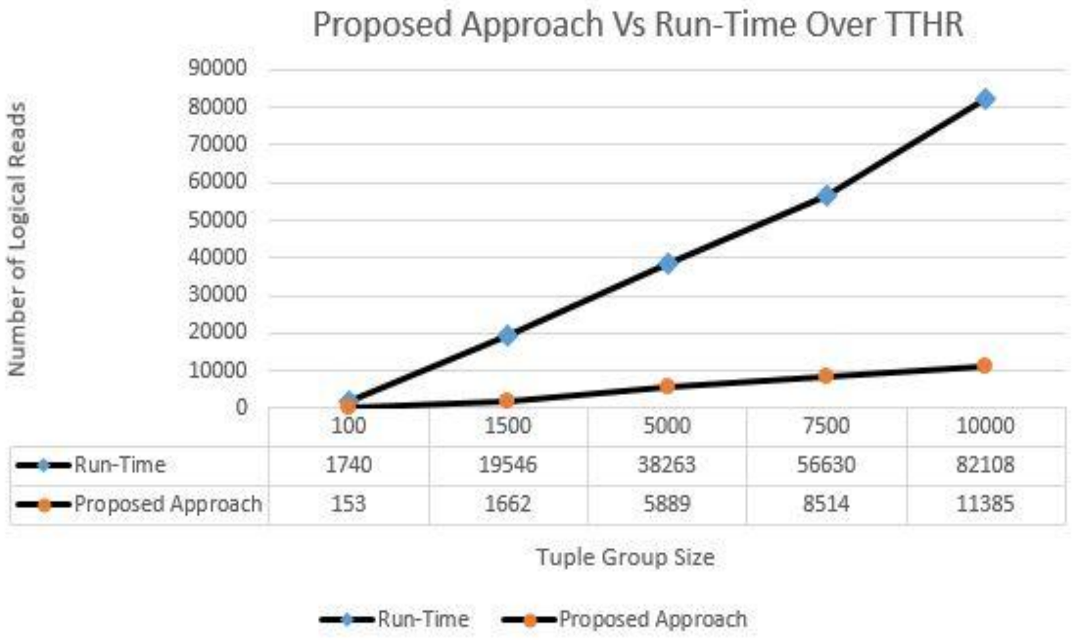
Fig. 4 and Fig. 5 show the comparison between proposed and run-time approach for aggregation on the basis of logical reads. Tuple group size represents the number of tuples present in a group. In a tuple group there is one identical tuple for each interval in a timestamp. Each tuple in the group shares the similar time invariant attributes, but has a different time intervals.



	100	1500	5000	7500	10000
Run-Time	1210	14358	33402	42355	78027
Proposed Approach	100	1500	5000	7500	10000

Figure 4: Logical reads over tuple timestamp multiple historical relation data model

Number of logical reads directly depends on the structuring of the underlying temporal data model. It means that the distribution of value attributes over the temporal relations play significant role. In tuple timestamp historical relation data model, all the value attributes are kept in single temporal relation [10]. On the other hand, all the time variant attributes are dispersed in separate relations in tuple timestamp multiple historical relation data model. It clearly shows that dispersion of attributes over the relations puts significant impacts on various parameters like memory cost, number of logical reads and query execution time.



	100	1500	5000	7500	10000
Run-Time	1740	19546	38263	56630	82108
Proposed Approach	153	1662	5889	8514	11385

Figure 5: Logical reads over tuple timestamp historical relation data model

Fig. 6 and Fig. 7 shows the comparison between proposed and run-time approach for aggregation on the basis of query execution time. Optimal query execution plan assigns the query optimizer a responsibility to transform the queries into an equivalent query and the time elapsed in execution of this query is grasped as query execution time. Comparison of the proposed approach with the run-time aggregation is performed by using time point or range queries. For each of the query, execution time is recorded after executing it ten times and then their mean value is considered for the purpose of comparison. PgAdmin tool provides the query execution time for the sample queries.

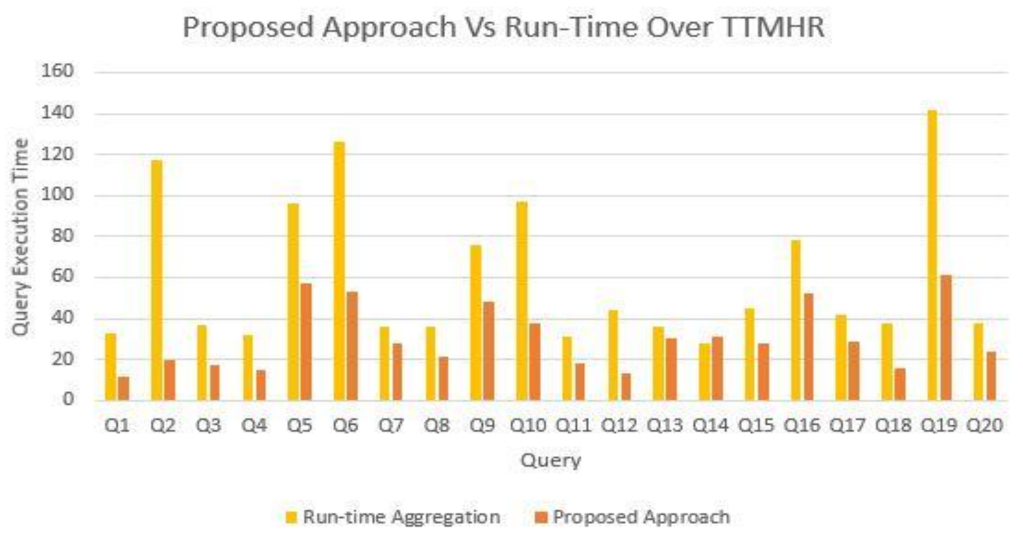


Figure 6: Query execution time over tuple timestamp multiple historical relation data model

Results of the experiment clearly reflects that the performance of the proposed approach for aggregation depends upon the degree of heterogeneity amongst the value attributes. The structure of the underlying schema directly influences the relationship existing amongst the attributes.

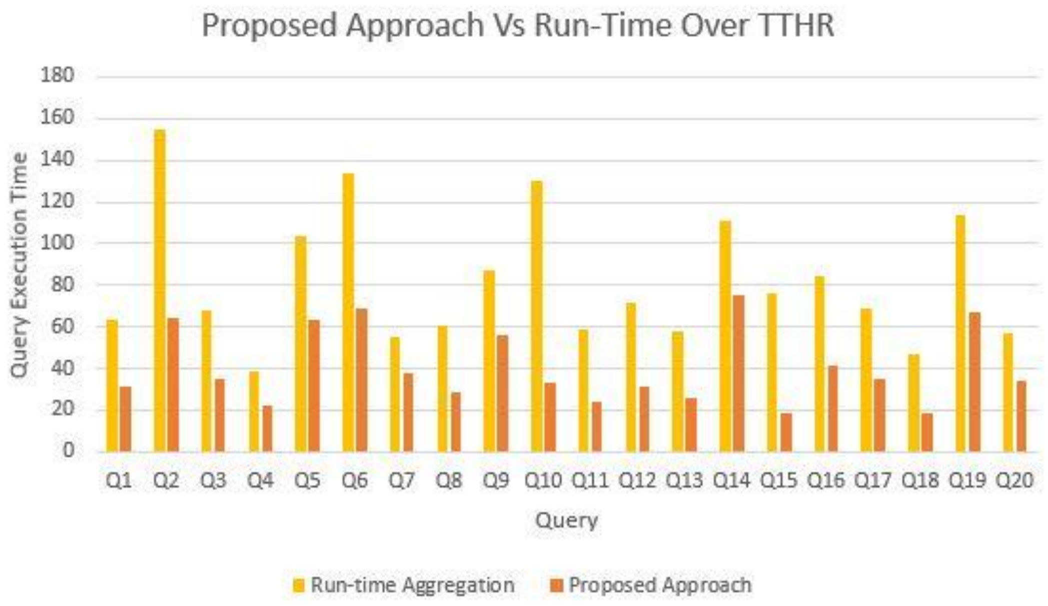


Figure 7: Query execution time over tuple timestamp historical relation data model

CONCLUSION

Temporal aggregation is an indispensable procedure in plentiful of time oriented data models as time oriented queries computed over non-aggregated data might yield malign results. Aggregation syndicates the temporal ranges of overlapping, adjacent and nearby value equivalent tuples. Since aggregation is an exorbitant process, aggregation of tuples is carried out while data is being updated to abate the query execution time. The aggregated result sets are stockpiled separately on the basis of extent of coupling as well as degree of heterogeneity amongst the time varying attributes. B-Tree indexing is furthermore applied on every result set to optimize the query execution time. This paper presents a comprehensive strategic flow to perform aggregation of value equivalent tuples during data update. It involves identification of veiled endpoints of an interval. Temporal predicates and temporal constructors are also employed for aggregation of partially aggregated result sets that, in effect, have the identical semantics as those of fully aggregated result sets. The significant discernment for most of the temporal predicates and temporal constructors is just to know about the un-veiled endpoints. We developed an index to store the partially coalesced points. A chunk of information is evaluated during triggering of an event and the results are stored separately. The technique used in this paper captures a lot of redundantly veiled points. A special index to store the veiled points should be able to reduce the storage cost of the veiled relation and speed-up the evaluation of temporal predicates and temporal constructors on partially coalesced timestamps. Finally, we tested our proposed approach by employing numerous temporal predicates in structured query language and testing those predicates using postgres. The results of the experiments clearly indicates that the performance of the proposed approach is much better as compared to aggregation at run-time. In future, we plan to extend partial aggregation to multiple time dimensions.

REFERENCES

[1] Aljawarneh, Shadi A., RadhakrishnaVangipuram, Veereswara Kumar Puligadda, and JanakiVinjamuri. "G-SPAMINE: An approach to discover temporal association patterns and trends in internet of things." Future Generation Computer Systems (2017).

[2] Carafoli, Luca, Federica Mandreoli, Riccardo Martoglia, and Wilma Penzo. "Streaming Tables: Native Support to Streaming Data in DBMSs." IEEE Transactions on Systems, Man, and Cybernetics: Systems (2017).

[3] Cheng, Kai. "Approximate Temporal Aggregation with Nearby Coalescing." In International Conference on Database and Expert Systems Applications, pp. 426-433. Springer International Publishing, 2016.

[4] Ding, Weilong, Shuai Zhang, and Zhuofeng Zhao. "A collaborative calculation on real-time stream in smart cities." Simulation Modelling Practice and Theory, 72, pp. 23-33. 2017.

[5] Dyreson, Curtis E. "Temporal coalescing with now granularity, and

incomplete information." In Proceedings of the 2003 ACM SIGMOD international conference on Management of data, pp. 169-180. ACM, 2003.

[6] Freytag, Johann Christoph, and Nathan Goodman. "Translating aggregate queries into iterative programs." In VLDB, vol. 86, pp. 25-28. 1986.

[7] Gendrano, Jose Alvin G., Bruce C. Huang, Jim M. Rodrigue, Bongki Moon, and Richard T. Snodgrass. "Parallel algorithms for computing temporal aggregates." In Data Engineering, 1999. Proceedings., 15th International Conference on, pp. 418-427. IEEE, 1999.

[8] Gordevičius, Juozas, Johann Gamper, and Michael Böhlen. "Parsimonious temporal aggregation." The VLDB Journal—The International Journal on Very Large Data Bases, 21(3), pp. 309-332. 2012.

[9] Guzzo, Antonella, Andrea Pugliese, Nino Rullo, DomenicoSacca, and Antonio Piccolo. "Malevolent Activity Detection with Hypergraph-Based Models." IEEE Transactions on Knowledge and Data Engineering (2017).

[10] Halawani, Sami M., Ibrahim AlBidewi, Ab R. Ahmad, and Nashwan A. Al-Romema. "Retrieval optimization technique for tuple timestamp historical relation temporal data model." Journal of Computer Science, 8(2), p. 243. 2012.

[11] Kaufmann, Martin, Peter M. Fischer, Norman May, Chang Ge, Anil K. Goel, and Donald Kossmann. "Bi-temporal timeline index: A data structure for processing queries on bi-temporal data." In Data Engineering (ICDE), 2015 IEEE 31st International Conference on, pp. 471-482. IEEE, 2015.

[12] Kline, Nick, and Richard T. Snodgrass. "Computing temporal aggregates." In Data Engineering, 1995. Proceedings of the Eleventh International Conference on, pp. 222-231. IEEE, 1995.

[13] Kumar, Shailender, and Rahul Rishi. "Retrieval of Meteorological Data using Temporal Data Modeling." Indian Journal of Science and Technology, 9(37), 2016.

[14] Kvet, Michal, Karol Matiako, and MarekKvet. "Managing and Storing Function Results in Temporal Approach." In Open and Big Data (OBD), International Conference on, pp. 87-94. IEEE, 2016.

[15] Lopez, IF Vega, Richard T. Snodgrass, and Bongki Moon. "Spatiotemporal aggregate computation: A survey." IEEE Transactions on Knowledge and Data Engineering, 17(2), pp. 271-286. 2005.

[16] Marcellino, Massimiliano. "Some consequences of temporal aggregation in empirical analysis." Journal of Business & Economic Statistics, 17(1), pp. 129-136. 1999.

[17] Moon, Bongki, I. Fernando Vega Lopez, and Vijaykumar Immanuel. "Efficient algorithms for large-scale temporal aggregation." IEEE Transactions on Knowledge and Data Engineering, 15(3), pp. 744-759. 2003.

[18] Navathe, Shamkant B., and Rafi Ahmed. "A temporal relational model and a query language." Information Sciences, 49(1-3), pp. 147-175. 1989.

[19] Radhakrishna, Vangipuram, P. V. Kumar, and V. Janaki. "Mining Outlier Temporal Association Patterns." In Proceedings of the Second International Conference on Information and Communication Technology for Competitive Strategies, p. 105. ACM, 2016.

[20] Snodgrass, Richard T., Santiago Gomez, and L. Edwin McKenzie. "Aggregates in the temporal query language TQuel." IEEE Transactions on Knowledge and Data Engineering, 5(5), pp. 826-842. 1993.

[21] Terenziani, Paolo. "Irregular indeterminate repeated facts in temporal relational databases." IEEE Transactions on Knowledge and Data Engineering, 28(4), pp. 1075-1079. 2016.

[22] Unnikrishnan, K., and K. V. Pramod. "On implementing temporal coalescing in temporal databases implemented on top of relational database systems." In Proceedings of the International Conference on Advances in Computing, Communication and Control, pp. 153-156. ACM, 2009.

[23] Yang, Jun, and Jennifer Widom. "Incremental computation and maintenance of temporal aggregates." In Data Engineering, 2001. Proceedings. 17th International Conference on, pp. 51-60. IEEE, 2001.

[24] Zhang, Donghui, Alexander Markowetz, Vassilis J. Tsotras, DimitriosGunopulos, and Bernhard Seeger. "On computing temporal aggregates with range predicates." ACM Transactions on Database Systems (TODS) 33(2), p. 12. 2008.

[25] Zhou, Xin, Fusheng Wang, and Carlo Zaniolo. "Efficient temporal coalescing query support in relational database systems." In International Conference on Database and Expert Systems Applications, pp. 676-686. Springer Berlin Heidelberg, 2006.

[26] Kumar, Shailender, and Rahul Rishi. "A relative analysis of modern temporal data models." In Computing for Sustainable Global Development (INDIACom), 2016 3rd International Conference on, pp. 2851-2855. IEEE, 2016.