Sensorical aspect of the temporal data architecture –pointer layer

Michal Kvet [1)], Karol Matiaško [2)]

1) University of Zilina, Univerzitna 8215/1, 010 26 Zilina, Slovakia, Michal.Kvet@fri.uniza.sk

2) University of Zilina, Univerzitna 8215/1, 010 26 Zilina, Slovakia, Karol.Matiasko@fri.uniza.sk

Abstract:

Development of information systems were and still is influenced by the approaches and techniques for data management. In the past, there was only small data amount processed, however, nowadays, the data portions to be handled, managed and stored is significant delimited by the validity, thus it is necessary to store also historical and future valid data. In the environment of sensorical data processing, the problem is really massive. To get optimal or sub-optimal solution performance, parallel and distributed architectures should be developed. In this paper, we deal with the grouping techniques based on developed pointer layer and highlight its impact on performance. All performance characteristics are based on column level temporal architecture, which is performance resistant to different granularities of the changes.

***Keywords*:** column level architecture, temporality, temporal distribution, grouping, pointer layer

1. INTRODUCTION

Development in any period of time is influenced by the data modelling methodologies and techniques. In the past, data were directly stored as a part of the application in user defined file structures, but with the need to process large data amounts. The challenges of their storing, effectivity and performance have been shifted to the database approaches. Paradigm of conventional database is based on actual data processing, however, nowadays, it is necessary to store also historical data and data, which will be valid in the future (e.g. planned reconstructions, repairs,…). Intelligent systems provide wide range and types of the data with specific characteristics and granularities of the changes [16] [17]. Also their reliability and precision is important factor. Whereas the data amount and storing requirements are still rising, real time and temporal databases have been developed to process and manage evolution and changes over the time. In the field of intelligent systems, data effectivity is really significant. When we look to the past, temporal characteristics have been proposed soon after the database system definition to ensure security of the stored data provided by backups and log files. However, these approaches are not suitable for large data management over the time and fast decision making [4] [5], therefore later temporal approaches have been proposed [1] [2] [3].

Standard temporal characteristics are based on extension of the primary key by the definition of the validity (uni-temporal system) or by using other attributes defining time limitations. It can be reflected by two attributes bordering start and end point of the validity or by just one attribute – start point. In that case, newer state should automatically delimit the validity of previous one state. However, whereas there is no support for period modelling in the database systems, consistency of the time validity must be checked by the user explicitly (e.g. end point must be higher or equal than start point, each object can be modelled by only one state during any timepoint or interval) – fig. 1.

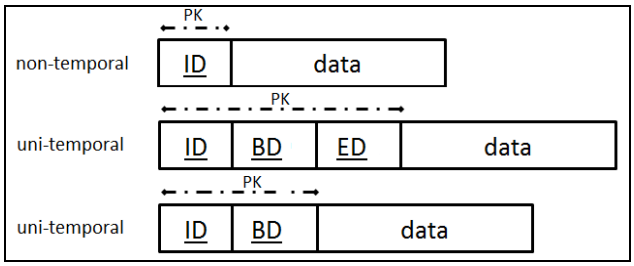

Fig. 1. Structure of temporal table [7]

In principles, we can use multi-temporal systems defining validity, transaction validity and other attributes delimiting time intervals [6].

2. COLUMN LEVEL ARCHITECTURE

The main disadvantage of previously defined temporal system is the effectivity, if not all attribute values are changed at strictly defined time. Whereas the whole state is changed, all attribute values must be stored, which brings many problems with storage efficiency and workload, as the system contains many duplicates. Moreover, delimiting individual attribute changes is also complicated, particular values must be compared each other to get information about the equality.

As a result it brought a significant drop in performance. Each table contains information from the specific location, each network node provides data from multiple sensors (often hundreds or thousands of sensors). The problem is that each sensor has a sensitivity and speed of the measurement and delivery of results. Assessment of the average processing time is not appropriate in terms of storage needs for managing unchanged values. Another factor is also potential loss of data. If we set the unit to second of time, and some of the sensors get data in microseconds, we lose one million of records that may contain important information and activities affecting system performance and response management. If the granularity is the smallest one, same values are still stored. Therefore, such temporal solution is inappropriate. Moreover, we have to store not only sensorical data, but also other instance parameters, settings, which evolve slowly, and changing them is rather rare or sometimes even impossible [8] [9] [10].

Transformation of temporal model based on the whole objects is not easy process. First of all, it must provide layer for dealing with existing applications managing actual or temporal data based on conventional or temporal approach. Thus, the first layer contains two access methods. The first one deals with only actual states in the object level form. The second method contains information about the evolution of the attribute values,

but provided in the form of object level architecture. These data are reflected and provided by the methods of views, so any change is registered automatically. However, *temporal management layer* uses attribute oriented granularity, which covers the second layer. Non-actual values of particular attributes are stored in the third values – historical values, but also values valid in the future. Changes can be planned and are executed automatically using *pointer layer*. Fig. 2 shows the architecture of our column level temporal approach. In this case, we use centralized architecture.

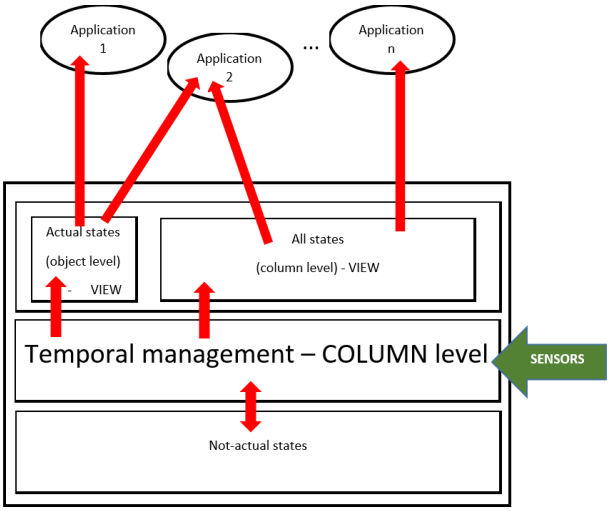

Fig. 2. Centralized column level temporal architecture

Internal column temporal level is characterized by the temporal table consisting of these attributes [9] [11]:

- *ID_change* – got using sequence and trigger – primary key of the table.
- *ID_previous_change* – references the last change of an object identified by *ID*. This attribute can also have *NULL* value that means, the data have not been updated yet, so the data were inserted for the first time in past and are still actual.
- *ID_tab* – references the table, record of which has been processed by DML statement *(Insert, Delete, Update, Restore)*.
- *ID_orig* - carries the information about the identifier of the row that has been changed.
- *ID_column* – holds the information about the changed attribute (each temporal attribute has defined value for the referencing).
- *Data_type* – defines the data type of the changed attribute:
  - *C = char / varchar, N = numeric values (real, integer, ...), D = date, T = timestamp, ...*

  This model can be also extended by the definition of the other data types like binary objects.
- *ID_row* – references to the old value of attribute (if the DML statement was *Update*). Only update statement of temporal column sets not *NULL* value.
- *Operation* – determines the provided operation:
  - *I = insert, D = delete, U = update, R = restore*

  The principles and usage of proposed operations are defined the in the part of this paper.
- *BD* – the begin date of the new state validity of an object.

Data table model is shown in fig. 3.

Temporal_table			
id_change	Integer	NN	(PK)
id_previous_change	Integer		
operation	operation_domain	NN	
id_tab	Integer	NN	
id_orig	Integer	NN	
id_column	Integer		
id_row	Integer		
bd	Date	NN	
data_type	data_type_domain		

Fig. 3. Temporal table in column level temporal approach [11]

Main part of the architecture and also bottleneck of the system is just *temporal management layer*. If the number of sensors is high, it does not have to cause problems. The most important part is based on the frequency and reliability of changes [12]. When moving to the milli or even micro or nano-seconds, such temporal solutions are insufficient. Moreover, if the whole state is updated at defined time, it causes one complex insert statement, however for object level architecture, and however, such defined solution reflects each attribute as separate statement. Concept of column level architecture is perfect for size requirements and provide suitable performance for the Select statements, however, how to keep the system fresh and powerful for long time period with many sensors and different granularity? The answer is to strengthen the system core part – *temporal management layer* by the means of parallel and distribute processing of different signals depending on definitions and approaches. Fig. 4 shows the extended temporal layer architecture dividing sensors into categories and grouping together to smaller blocks. Thanks to that, we do not have one complex table, but data are relocated into smaller portions to separate temporal tables, which are also modelled on the level of attributes.

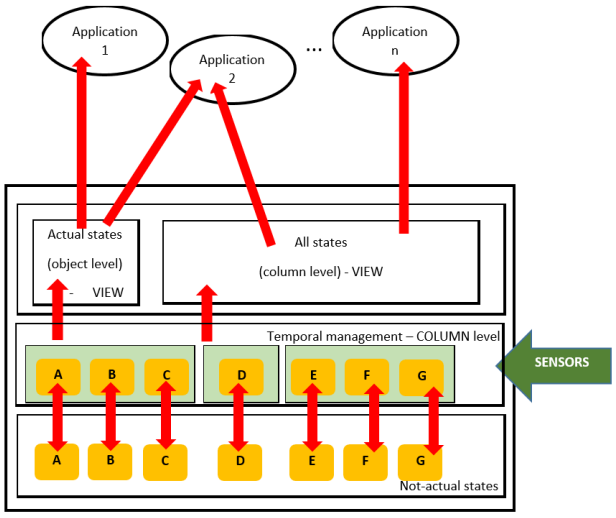

Fig. 4. Extended temporal layer architecture

Whereas actual states and object level states can be obtained only by the querying views, therefore, it is necessary to store them in temporal layer (second layer). In this case, we can extend that layer by management conventional (non-temporal) tables and even static tables, which values cannot be changed at all, like code lists (fig. 5).

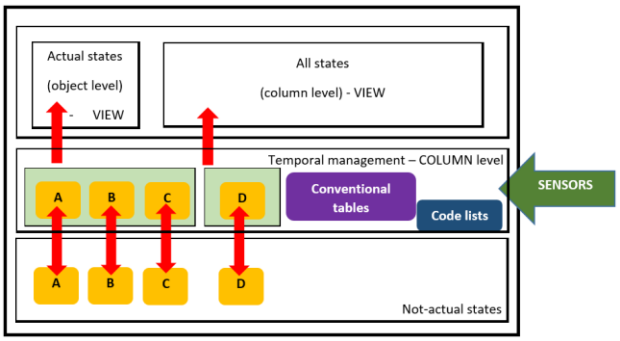

Fig. 5. Extended temporal layer architecture covering conventional tables and code lists.

In extreme case, each attribute can be transferred to separate table grouping layer to form the finest granularity – updating is effective, however, such solution provide really poor performance for other operations (*Select*) and also size of the whole structure due to blocking factors.
For the purpose of data modelling in multiple table locations (which can be even located in several servers), another internal layer has to be defined to provide particular data locality - to average workload of the nodes, group assignments can be relocated automatically over the time. Let the layer call *pointer* layer (fig. 6), which dispatches individual requirements to the particular group to be processed and to return defined and required results. Whereas assignments are managed automatically, several groups can be reported. This layer looks is defined between *view layer (1)* and *column level temporal layer (2)* and provides interface between them to detach external layer (which can be managed by the applications) and internal layer managing data using attribute oriented approach.

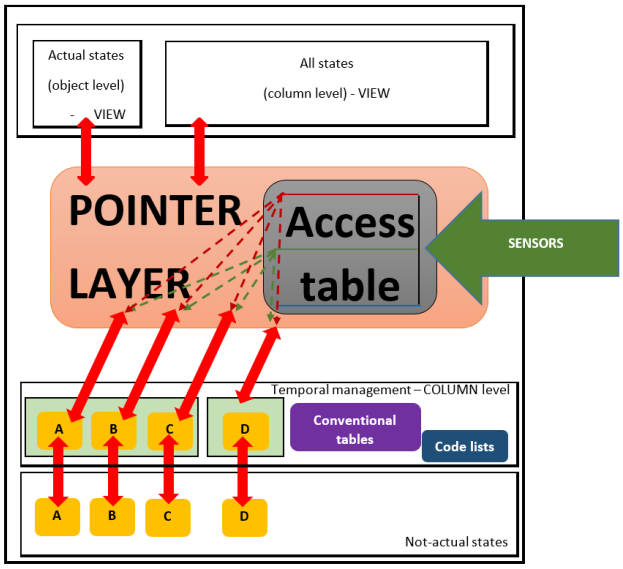

Fig. 6. Pointer layer architecture

As it has been mentioned, adding sensors to the group during the defined time interval is provided by the *pointer layer*, which must also manage individual assignment changes and monitor impacts of workload on global performance. Data assignments are stored in *access_table* with the following structure (fig. 7):

- **ID** – primary key of the table provided by the sequence and trigger,
- **BD** – delimits the begin timepoint of the assignment to particular group node,
- **ED** – stores information about the last point of the assignment to particular group node, can be *NULL*, which expresses actually unlimited assignment,
- **Group_id** – identifier of the group (foreign key) in the column temporal layer,
- **Sensor_id** – identifier of the sensor (foreign key).

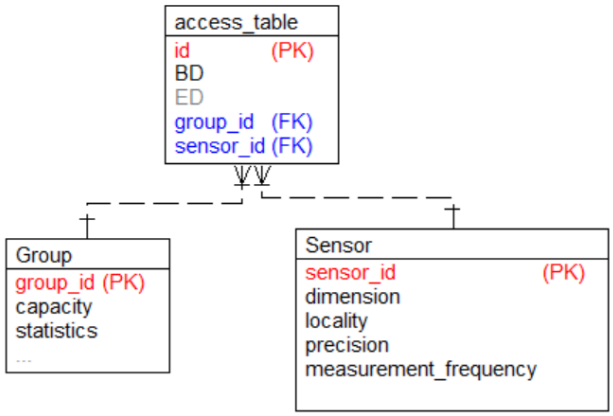

Fig. 7. Access_table

In the next part, it is necessary to evaluate the complex performance impact, when individual sensors are processed separately and model defined number of groups. Each category (group) is defined by a random assignment of the sensors with regards on the approximately the same workload for each group, thus it is necessary to minimize extremal values [13].

3. POINTER LAYER AND SENSOR GROUPING

The aim of the pointer layer is to provide management of the associations between groups and sensors themselves. It is necessary to evaluate data management performance. In intelligent systems reflected by temporal databases, it is modelled using state and attribute changes and process of getting required data during the defined timepoint or time interval. Column level temporal architecture statement types can be divided into four types:

- *Insert* statement, which expresses adding new sensor processing to the system.
- *Update* statement – operation providing changes of the attribute value based on sensor data measuring, processing and evaluating.
- *Delete* statement – operation for unregistering sensor processing caused by failure or removing sensor due to its unnecessity.
- *Purge* statement – although processing is based on time and temporal data should be stored over the time, usually it is not necessary, nor appropriate to store all values during infinite time period (*volatility*). These data are too extensive, but mainly, they are no longer needed for processing and then, they do not produce any effect for decision making, evaluating and setting parameters. Therefore, *Purge* method is

used to remove such data from the temporal system. Of course, in this case, they are transferred and moved to the data warehouse, thus they remain available for the needs of archives.

Thanks to that architecture, the most important data destructive operation is just *Update* statement, because *Insert* and *Update* statements can be executed during small server loads (e.g. at nights) and are usually executed rarely. On the other hand, if it is necessary to add new sensors immediately, it does not affect performance so significantly.

For the purposes of this paper, we will therefore deal with *Update* and *Select* statements.

4. PERFORMANCE EVALUATIONS

Our experiments and evaluations were performed using defined sensorical network environment – 1000 sensors were used delimited by 10 000 changes for each of them. Thus, total number of processed data is 10 000 000, which is, as we can see in the experiments, totally suitable for evaluations.

Experiment results were provided using Oracle Database 11g Enterprise Edition Release 11.2.0.1.0 - 64bit Production; PL/SQL Release 11.2.0.1.0 – Production. Parameters of used computer are: processor: Intel Xeon E5620; 2,4GHz (8 cores), operation memory: 16GB and HDD: 500GB.

Experiment results comparison was obtained using *autotracing* highlighting time [14] [15].

The first part of the experiments will deal with the number of the groups and column level temporal tables for sensor processing and its impact on the performance of selecting. We will manage and compare performance with regards and comparison with centralized architecture with only one node. For the evaluation, we will build solutions using 2, 3, 5, 10 and 100 grouping nodes. In case of retrieving too small data portion, performance is a bit worse (but only slight). The reason is based on *pointer layer*, which must evaluate and control access to individual grouping nodes. Whereas the time evolution can cause changes in the assignments, processing lasts some time, however, it reflects only tenths of seconds (approximately 0,1s – average value). On the other hand, another significant performance factor influencing processing is just splitted table management. Whereas such table contains significantly less data amount, such processing and obtaining desired data is really far faster. Thus, the impact of the pointer layer is minimized, even replaced by the improvement of subsequent data retrieval. To evaluate also characteristics of the pointer layer, one hundred assignments have been performed during each experiment to balance workload of each group. Principles of balancing based on statistics is shown in fig. 8. In the first case, three sensors are associated to the first group. However, the workload of this group is rising. If the difference of the workload across the groups is higher than 10%, rebalancing is started automatically. The aim is to have approximately the same workload. As a consequence, the third sensor is migrated to the second group. Now, the workload of the groups are balanced. If there is later any significant change, rebalancing is started automatically, too. Thus, the associations are delimited by the time interval.

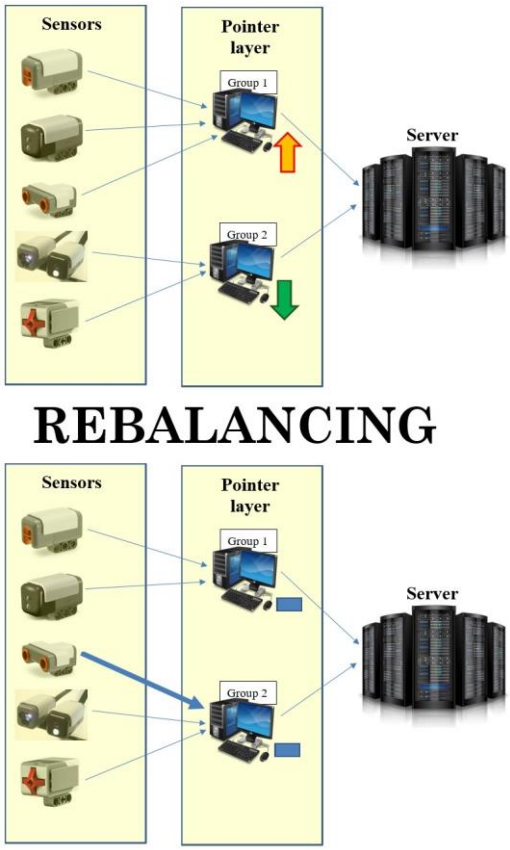

Fig. 8. Rebalancing

When dealing with centralized architecture, only one group is used and all input data are located there. As a consequence, there is significant data amount and workload management. Moreover, there have been also situations, when data were delayed and even lost, because input buffers were full for accepting more data requests. In the fig. 9, there is the diagram of the centralized data retrieval. We deal with various numbers of data in comparison with all of them stored in the table. As we can see, there is really significant processing time growth. When dealing with all the data, processing lasts more than 320 seconds. Even when retrieving only 1% of the data, processing costs reflected in time is approximately 3,44s, which is really high for huge data requirement amount over the time.

Therefore, we propose another layer managing all the splitted data table segments. Thanks to that, each segment contains smaller data amount and management, but mostly retrieval is easier reflected by particular indexes [xxx].

In this part, we have compared six grouping architectures, which are delimited by the number of grouping nodes. In the first phase, we have compared performance using 2, 3, 5, 10, 50 and 100 grouping nodes. The workload and data have been distributed randomly, but with emphasis on uniform distribution. As in the previous section, the assignment change number was 100. Fig. 10 shows the results in the table form. Thus, number of grouping nodes has huge performance impacts on processing time, although it must be extended by another pointer layer. Moreover, it has been performed on single

server, so there so no another impact based on simplifying workload to distribute requirements to nodes for distribute and parallel processing.

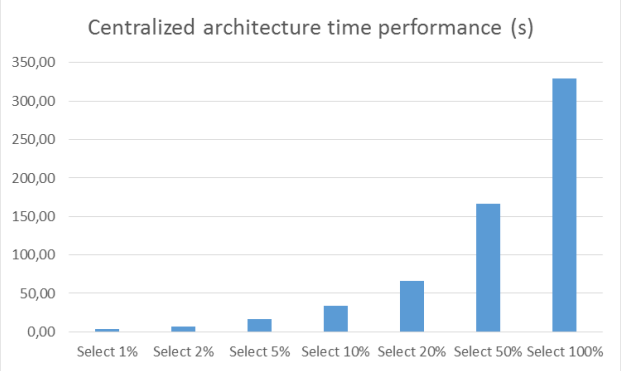

Fig. 9. Centralized architecture performance

When selecting only small data portions (1%), there is no improvement or very slight. On the other hand, when dealing with more data portions, processing can be distributed and pre-processed by the grouping node. Thanks to that, when dealing with 100 nodes, there is even 99% performance improvement, so the processing time descended from the values 329,34s to 3,26s due to parallelism. Moreover, when dealing with only one server, it is not necessary to deal with node failures, communication network faults, etc.

number of nodes	Select 1%	Select 2%	Select 5%	Select 10%	Select 20%	Select 50%	Select 100%
1	3,44	6,67	16,92	33,37	66,10	166,13	329,34
2	3,58	6,75	16,76	34,14	67,64	168,96	168,87
3	3,40	6,83	16,60	33,69	67,22	111,06	111,06
5	3,21	6,46	16,23	32,58	66,27	66,43	111,06
10	3,01	5,52	13,84	24,73	27,86	27,45	27,88
50	3,22	5,50	6,50	6,93	7,12	7,89	8,30
100	3,30	3,40	3,72	3,97	4,34	4,99	5,63

Fig. 10. Experiment results – processing time (s) based on number of grouping nodes.

Fig. 11 shows the time processing, when dealing with 50 and 100 nodes. It is also reflected by the number of data to be processed and retrieved. It is compared with the centralized architecture, so only one node is used and pointer layer can be therefore completely omitted.

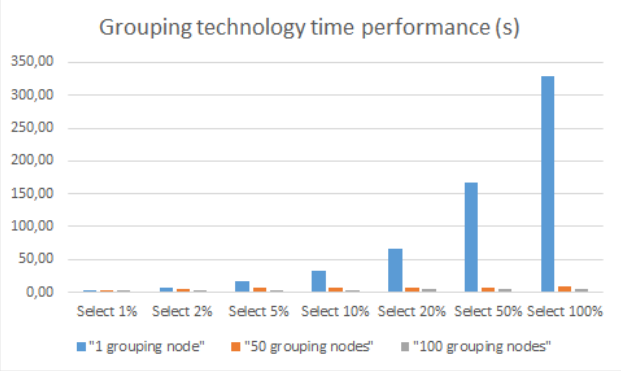

Fig. 11. Experiment results – processing time (s) based on number of grouping nodes and centralized architecture.

Data retrieval is, however, only one part of the problem. Another important performance factor is just based on changes management operated by *Update* statements. Although it is perfect for data selecting, it is inevitable to optimize and manage also mentioned operation. For this purpose, we have evaluated *Update* statement performance for centralized architecture, but also for 10 and 100 grouping nodes. Fig. 12 shows the experiment results. Individual time performance characteristics are shown in fig. 13 (centralized architecture), fig. 14 (10 nodes), and fig. 15 (100 nodes). Each *Update* statement type has been performed 1000 times and shown results express summed values.

Comparison of the column level temporal architecture in comparison with object level is described and experimented in [xxx] [xxx]. *Update* statement itself can be divided into two separate operations, which must be executed in temporal system. Whereas it is necessary to ensure correctness of the results reflected by the unique state during each timepoint, *Select* statement must provide control mechanisms. Then, update of the state is provided. Therefore, the main part is based on controlling mechanisms, which are provided by the *Select* statement and result checking and managing. Whereas we have 1000 sensors, we will deal with various characteristics, in this phase, we deal with 1%, 2%, 5%, 10%, 20%, 50% and 100% of all sensors in one operation. As we can see, if groups are defined, significant improvement is always provided. For updating only 1% of all sensors, when dealing with 10 grouping nodes, performance is better and time processing is lowered by 48,22% for 10 nodes and even by 67,22% for 100 grouping nodes (in comparison with centralized architecture - reference 100%). Moreover, when all sensors are updated during the same time, comparing time processing between grouping nodes and centralized architecture (reference 100%) provides following results:

- *51, 25%* 10 grouping nodes,
- *95, 36%* 100 grouping nodes.

Update	centralized architecture	10 nodes	100 nodes
1%	2,53	1,31	0,82
2%	4,44	3,05	1,59
5%	10,64	7,57	4,12
10%	18,12	15,12	7,43
20%	40,43	29,14	8,42
50%	86,04	75,18	12,24
100%	312,65	152,43	14,52

Fig. 12. Experiment results – processing time (s) based on number of grouping nodes.

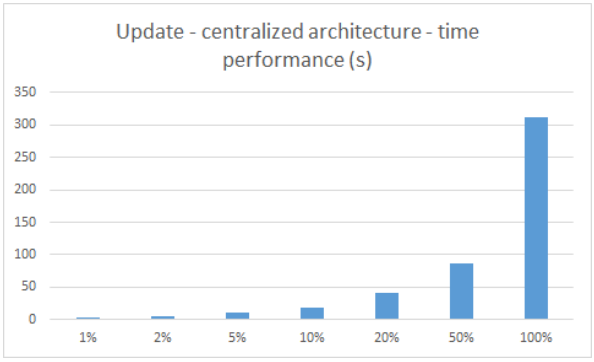

Fig. 13. Experiment results – processing time (s) of the Update statements – centralized architecture.

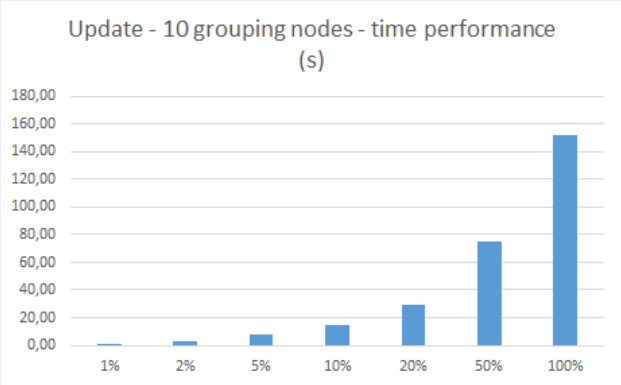

Fig. 14. Experiment results – processing time (s) of the Update statements – 10 grouping nodes.

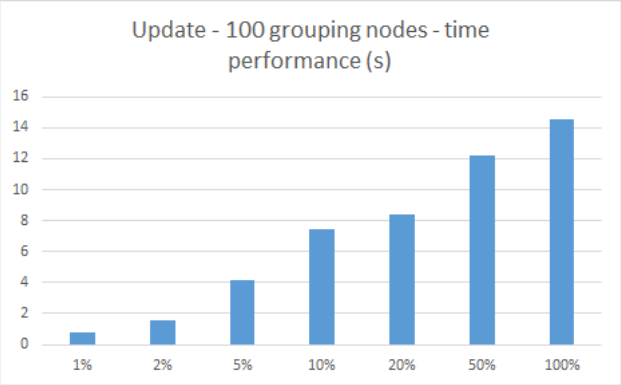

Fig. 15. Experiment results – processing time (s) of the Update statements – 100 grouping nodes.

5. CONCLUSION

Sensor data processing, managing and evaluating is one of the significant part of the database technology. They provide data over the time to reflect changes and evolutions. Such data are complex and should be stored in the database. Therefore, it is necessary to design and propose effective and robust solution to provide desired performance. Paradigm of the conventional database is not suitable at all, because it is based on managing only actual valid data. Temporal management defined in the past is based on object level granularity, thus, such solution does not provide sufficient power for sensor data processing, and therefore we have proposed column level temporal architecture, which consists of three layer levels. Whereas the workload and onslaught requirements for the column level temporal layer is enormous, it is necessary to extend processing and management. In our case, we have extended it by the pointer layer, which stores associations of the sensor to the particular group, which can evolve over the time based on statistics and workloads. Based on defined experiments, such solution provides significant improvement in the process of data retrieving, managing and changing.

In the future, we would like to extend that solution by the replications to the multiple servers with various number of grouping nodes.

6. ACKNOWLEDGMENT

This publication is the result of the project implementation:

Centre of excellence for systems and services of intelligent transport, ITMS 26220120028 supported by the Research & Development Operational Programme funded by the ERDF and *Centre of excellence for systems and services of intelligent transport II.*, ITMS 26220120050 supported by the Research & Development Operational Programme funded by the ERDF.

"PODPORUJEME VÝSKUMNÉ AKTIVITY NA SLOVENSKU
PROJEKT JE SPOLUFINANCOVANÝ ZO ZDROJOV EÚ"

7.REFERENCES

[1] L. Ashdown. T. Kyte *Oracle database concepts*, Oracle Press, 2015.

[2] C. J. Date. *Date on Database*, Apress, 2006.

[3] S. Feueuerstein. *Oracle PL/SQL Programming*, O'Reilly, 2014.

[4] J. Janáček. M. Kvet. *Min-Max Optimization Of Emergency Service System By Exposing Constraints*, in Communications: Scientific Letters of the University of Žilina, volume 2/2015, 2015, pp. 15 – 22

[5] J. Janáček. M. Kvet. *Public service system design by radial formulation with dividing points*, in Procedia computer science Vol. 51, 2015, pp. 2277 – 2286

[6] T. Johnston. *Bi-temporal data – Theory and Practice*, Morgan Kaufmann, 2014.

[7] T. Johnston. R. Weis. *Managing Time in Relational Databases*, Morgan Kaufmann, 2010.

[8] M. Kvet. K. Matiaško. *Temporal Context Manager*. 2015. SDOT Žilina, pp. 93-103.

[9] M. Kvet. K. Matiaško, *Transaction Management*. 2014. CISTI, Barcelona, pp.868-873.

[10] M. Kvet. K. Matiaško, *Uni-temporal Modelling Extension at Object vs. Attribute Lev*el, 2013. IEEE conference EMS 2013,

[11] M. Kvet. M. Vajsová. *Transaction Management in Fully Temporal System*, 2014. UkSim, Pisa, pp. 147-152.

[12] T. Kyte. D. Kurn. *Expert Oracle Database Architecture*, Apress, 2014.

[13] K. Matiaško, et al. *Database systems*. EDIS, 2008.

[14] H. Molina, et al. *Database systems* – The complete book", Pearson, 2008.

[15] R. Rood, et al. *Oracle Advanced PL/SQL Developer Professional Guide*, Packt Publishers, 2012.

[16] P. Vilhan and J. Gajdoš, Tool for Rapid Acceleration of Network Simulation in OMNeT++", 2012. In UkSIM 2012, Cambridge.

[17] P. Vilhan. L. Hudec. *Building public key infrastructure for MANET with help of B.A.T.M.A.N advanced*, 2013. In EMS 2013, Manchester.