Using multi-index database layer in ad-hoc communication networks

Michal Kvet[10000-0003-3937-7473] and Martina Durnekova[2]

[1,2] Zilinska univerzita v Ziline, Univerzitna 8215/1, 010 26 Zilina
Michal.kvet@fri.uniza.sk

Abstract. Current wireless communication in ad-hoc networks requires a sophisticated approach for the data location, retrieval, and transfer, to use the connection availability appropriately. Data are commonly stored in the databases spread across the network or located in each node. This paper deals with the current techniques of data location and retrieval using the index set. The main limitation is the availability and suitability of the index for the particular query request. Our proposed solution is based on the multi-indexing layer, by which the several indexes can be used to a single query in parallel, followed by the merging operation using bitmap index mapping. Thanks to the proposed architecture, data location and efficiency of the whole system can be ensured. The robustness and scalability of the solution can be reached, as well.

Keywords: ad-hoc network, data retrieval, multi-indexing layer.

1 Introduction

Ad-hoc networks are now widespread almost in any sphere. Communication is done by applying and identifying individual sources, which can produce or consume data dynamically. In intelligent transport systems, the significant requirement to ensure reliability, suitability, and relevance identification can be felt. Produced data amount is still rising, however, the quality and reliability of the data cannot be generally guaranteed. During the communication and data transfer, the total sent data amount is strictly limited by the time frame, individual elements are in the ad-hoc network. Thus, it is necessary to optimize the data stream, to identify the most important data objects and values to be transferred. Data themselves must be optimized in two manners. Firstly, relevant data should be identified to be transferred based on the temporal and spatial manner. Only data, that have not been transferred to the destination element should be handled. The second performance aspect is associated with the data location, by identifying data tuples physically in the database, by accessing data blocks, where individual tuples resist. Such provided data are then evaluated based on reliability and used as input for consecutive decision making [4].

This paper deals with the performance of the data tuple identification using index structures. Data are commonly stored in relational databases with limited disc and memory capabilities, thus the loading operation should be optimized, as well. Physi-

cally, individual tables consist of the segments (structure definition) and extents organized as a set of blocks. Data tuples are stored in the fixed-size blocks, which must be loaded into the memory for the evaluation. Memory Buffer cache structure is formed by the block matrix. I/O operation is always done by using block granularity. Sequential scanning naturally does not bring any benefit, whereas it would be too time and resource-demanding. Whereas ad-hoc network communication is short-term, the aspect of time is crucial. Index structures can navigate the processing in an optimized manner by transforming the linear complexity to the logarithmic (O(n)=> O(log(n))). This paper aims to propose a new multi-indexing layer, whereas the data demands can be dynamic and can even evolve. Thanks to that, any requirement can be passed by the index repository to optimize the data location.

This paper is organized as follows: Section 2 deals with the state of the art, it summarizes the principles of current relational database access principles with emphasis on the access structures and reliability of the index covering. In section 3, our proposed multi-index solution is defined. The performance of the proposed solution in comparison with conventional techniques is in section 4.

2 State of the art – index location and optimization

Data retrieval is a complex process starting in the user session, by transferring the query request to the server process, which is then processed and evaluated by the instance background processes implementing data identification and location. Database optimizer is there inevitable, whereas it provides the decision dealing the physical data access.

Query evaluation and data location is a staged process. Fig. 1 shows the core processing model of relational data management. Parser performs syntactic and semantic query analysis by planning transcription of the original query into a set of relational algebra operations. Optimizer gets the most efficient way to get the result of the query. The decision is done based on the optimization methods, available statistics. It selects the best suboptimal query execution plan based on the heuristical approach. Row Source Generator is the input of the suboptimal plan created by the optimizer. It creates the execution plan for a given query in form of the tree evaluator. The last step is the execution itself, according to the execution plan.

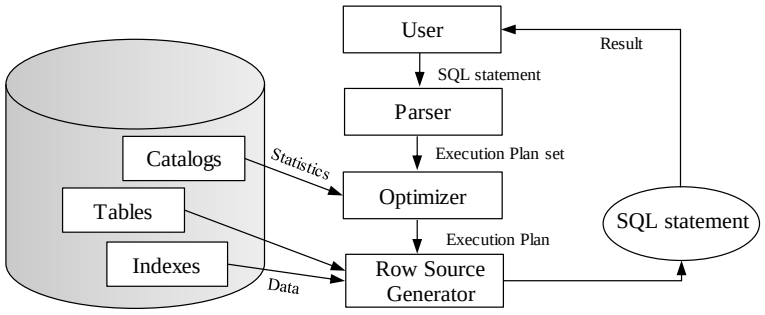

Fig. 1. SQL statement processing

For the evaluation and performance optimization treated in this paper, the most relevant operation is just the Optimizer stage, which covers the index usage decision. The index is an optional structure associated with the database table, by which the data tuples can be easier located in the physical storage. Various structures can be used, like B+tree, hash, bitmaps, etc. [2] [6], however the most relevant and most often used type is just the B+tree formed by the root node, internal elements, and leaf layer consisting of the pointers to the data – ROWIDs.

ROWID is an address to the physical database locating the data file, data block, and position of the row inside the block. By using the Table Access by Index Rowid (TAIR) method, data can be obtained optimally in terms of the data access time [2].

Conventional B+tree index type consists of the set of attributes with reflection to the attribute order inside, whereas it shapes the format. Namely, the order of the attributes or function call results delimits the structure of the index, the first element is the most significant reflecting the Where condition of the query. Index suitability evaluated by the optimizer is based on the covering Where conditions of the query followed by the table interconnection and Select statement clause. Optimally, all data attribute values are located inside the index, so the database itself does not need to be accessed. If not, ROWID is used to pass it through the TAIR method. There are, however, many limitations of the conventional index set in ad-hoc networks. Firstly, it does not reflect the importance of data reliability. Whereas the undefined values cannot be mathematically compared [2], they are not part of the index set, thus if the reliability aspect is to be treated, the whole table needs to be scanned sequentially [6]. It can be partially solved by indexing function results, by which the undefinition is limited [6]. It brings, however, additional storage and representation demands [5]. In the following syntax, three table attributes are indexed, delimited by the data transformation using NVL functionality converting NULL value into a specific denotation:

Create index ind on T(nvl(A, ‘x’), nvl(B, ‘x’), nvl(C, ‘x’));

NULL_MODULE has been introduced in [5]. It uses a specific structure outside the main index, where the undefined values are located. The main index just has a pointer list to the separate module. Thanks to that, it is ensured, that all data portions can be always located by the index physically.

Create index ind on T(A,B,C NULL_MODULE);

In ad-hoc networking, it is not just about the reliability and NULL value location and representation, an important aspect plays the role of the suitability, but mostly processing time and data location process, which can be delimited by various dynamic parameters, like occurrence time, durability, spatial positions, importance, etc. [1] [2] [9]. These aspects should be handled by the index set, as well as the direct attribute value representation or pointing respectively. As stated, the structure, data amount, and attribute set specification are dynamic and are defined and influenced by the destination modules. Therefore, the conventional index set delimited by the fixed attribute set list is not suitable. Namely, if the order of the attributes inside the index is

optimal, Index Unique Scan or Range Scan is used, based on the unique aspect of the provided data. Vice versa, if the order of the attributes is not suitable reflecting the provided query, either full index is scanned to obtain data (Full index scans methods) [9] the leading attribute of the index is skipped from the evaluation (Index skip scan) [2] [9].

In conventional systems, there is the requirement to balance the index directly in the main transaction ensuring the traverse path from the root to any leaf node to have the same length. In [5], a discussion about the index balancing strategy is present. The solution presented there is based on the Notice list, by which the index balancing operation is extracted into the separate transaction operated internally by the Balancer background process. Based on the evaluation, it can bring additional benefits of the data insert, even up to 50% for the processing time [5]. Select operation additional demands are approximately 11% based on the experimented environment [5].

When dealing with ad-hoc network communications, the time for the data providing and data exchange is strongly limited, thus the transfer process, as well as the data location should be strictly optimized to maximize the information values of the processed and obtained data. In this paper, we propose a multi-indexing layer, which removes the limitation of the already existing solutions. As stated, the order of the attributes for the evaluation is significantly influencing the access method and processing demands, and time costs. Our proposed solution uses dynamic indexing with multiple layers interconnected. In comparison with the auto-indexing presented in [10] introduced for the DBS Oracle database system, the index set is not changed over time, whereas we assume, that the system is so dynamic, that the costs for the index construction would lead into the process merging with no relevant output – the time to create new index is too high in comparison with the available time for the communication itself.

3 Solution

As stated, in the ad-hoc network, the efficiency of the data retrieval and transfer is a crucial process delimiting the whole system's usability. Conventional index set supervised by the administrator analyzing workload does not bring suitable benefit, whereas it is too dynamic. Similarly, auto-indexing technology supervised by the autonomous database in the Cloud environment is not feasible mostly due to the high rate of structure and rate changes [3]. Therefore, in this paper, we analyze the impacts and benefits of multi-layer indexing to ensure performance.

3.1 Monitored aspects

The conventional database is delimited by storing only current valid data, by replacing the original state with the new version, if the Update operation is executed. Although original data tuple can be found in the transaction logs temporarily, it is not part of the system architecture, it is not part of the core database, thus the process of historical data evaluation would be too demanding, whereas all transaction logs would be necessary to be scanned and evaluated sequentially. Temporal database architec-

ture extends the perspective by covering all temporal dimensions in the core system, either located in the main table [2] (by extending the object definition itself) or by proposing multi-level architecture [6]. One way or another, intelligent information systems and ad-hoc communication networks always need to store data temporally with the reflection to the historical data, current valid image, as well as future plans and perspectives. The first covered aspect is temporality, which can be modeled using three dimensions (history, current states, and future) with various granularity aspects (object [2], attribute [6], group [7]). The second covered aspect is just the spatial dimension modeled either by the positional data or by the affiliation to the region. The region is commonly based on space differentiation and allocation to detect future space of interest. Note, that generally, it can be denoted dynamically based on current conditions and can evolve. The third aspect is expressed by the data quality. Several undefined values can be present in the system, which can be evaluated. NULL value representation itself does not endorse the problem complexly, whereas it can originate from various reasons, like delayed value, the value provided from non-secure source, value out of expected range, non-verified data tuple provided from the new element, which has not been tested, yet, etc. In the proposed solution, the original NULL value is replaced by the set of reasoning values, which are covered by the separate module located in the instance memory (NULL_ORIG_MODULE) of the static node, to which individual states can point. This module is started during the mounting process of the instance loading and is resistant during the instance state is OPEN. Physically, the definition is stored in the data dictionary and pointed by the control file. It can be created and associated with the following commands:

Alter system create NULL_ORIG_MODULE scope={memory | spfile | both};

NULL_ORIG_MODULE can be created directly in the running instance, or specified in the instance parameter file (spfile), or can be applied in both systems directly. Note, that if the spfile option is used, a particular module will be launched just after the instance restart.

Then, individual NULL value meaning and representations can be registered:

Alter NULL_ORIG_MODULE
{add | remove | stop} null_type_identifier [meaning]
scope={memory | spfile | both};

NULL_ORIG_MODULE element has an associated integer value as the identifier, which can be specified either automatically (for the create (add) operation) or manually for other operations. It offers three option usage – to add a new node with the specified meaning, to remove an already existing element, or to stop it temporarily. Remove operation of the existing node can be enhanced by three options – restrict, remap, dangling. Restrict option does not allow to remove the element if at least one state points to that. Internally, several occurrences for each type identifier are stored in the data dictionary, so the system can easily evaluate, whether it is possible to drop it or not. Remap option takes all of the pointers, which are associated with the element

to be dropped, and remaps them to another existing element part of the NULL_ORIG_MODULE. Finally, the Dangling option of the undefinition aspect removes the element from the system, so the existing pointers are still there but are not associated with any element expressed by the dangling state.

NULL_ORIG_MODULE architecture is shown in fig. 2. Database instance is responsible for the NULL_ORIG_MODULE creation as part of the instance memory, supervised by the NULL_MANAGER background process. Thanks to that, original NULL values are replaced and can be part of the indexes as a pointer layer. It is located outside the main index, interconnected to the root index element.

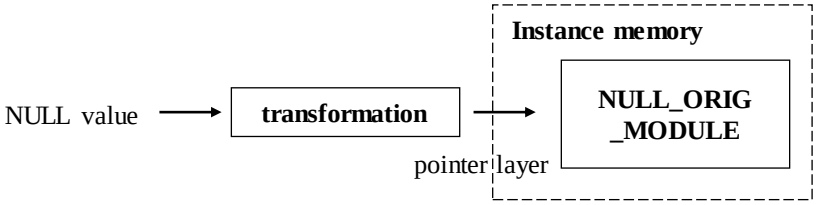

Fig. 2. NULL_ORIG_MODULE

3.2 Database dimension

The temporal database is a widespread solution dealing with state monitoring over time. Various granularity perspectives can be used. In object-level temporal architecture, object definition itself is extended by the time positional data expressing the time frame. If just one temporal dimension is used, the created model is uni-temporal. In our proposed solutions dealing with the un-trusted communications inside the ad-hoc network, the three-dimensional temporal model is used to identify individual states. Validity expresses the time frame, during which a particular data tuple was identified as valid, defined by the associated attribute values. The second time element reflects the transaction time, which delimits the original Insert operation to the whole system (global operation). The third timestamp expresses the local operation – record acquisition time during the data transfer. Optionally, the proposed solution can be extended to manage the processing time, as well, whereas the received data do not need to be evaluated immediately, but can be shifted into the input queue. In such a case, the delay between received time and processing time can be handled and consecutively optimized.

In the above principles, there is an assumption, that the provided data are relevant and reliable. Undefined values are expressed by the pointer denotation. Honestly, it is not possible to update the existing state as a correction, whereas the original value would be replaced. To propose a robust architectural model, another time element has to be introduced covering the versioning. In principle, it is always true, that the current object state version is always considered correct. Later, however, this version may be replaced by a new image. Therefore, there is an assumption, that the last inserted version is considered to be correct at the given time. In the ad-hoc network environment, individual nodes can hold irrelevant data tuples, which were later corrected. In such a case, the communication can be bi-directional and even the destina-

tion node can invalidate data from the provider if it holds newer data versions. Fig. 3 shows the principle of tuple identification. In the first phase, object versions are identified, expressed either based on the time point of the data origin or the sequence number, which must be, however, synchronized across the whole network, otherwise, the reliability issues can be identified. Data transfer is done only if the newer data version can be provided [7].

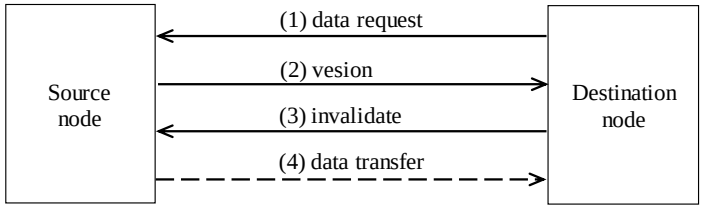

Fig. 3. Data transfer – reliability and usability

3.3 Architecture

The architecture of the whole solution is based on multiple interfaces. It consists of several internal interfaces, which can be categorized. The first type is based on time elements specified in 3.2. For each granularity representation, a separate interface is present. The second category covers the spatial definition, similarly modeled by two streams – direct positions and region association (reflecting the time perspective). The third module covers the undefinition by the interface. And finally, the last category is covered by user-defined access types to locate relevant data. The architectural model is in fig. 4.

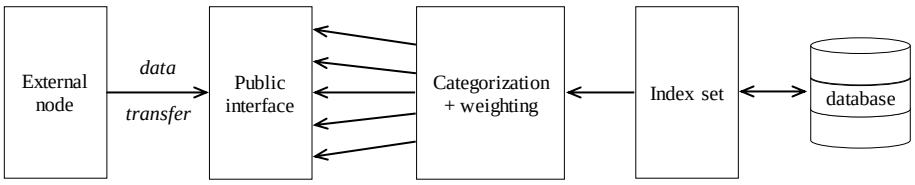

Fig. 4. Architecture - interface

Physically, the implementation is done by the group level temporal architecture, by which the synchronized groups are processed as a single group (attribute), so the storage demands are lowered – the temporal layer covers the whole group, not individual attributes separately.

3.4 Indexing

As already stated, multiple spheres covered by the spatial attributes, temporality, and user definitions are present. For each element, an index structure is present. The principle of the relational system indexing supervised by the B+tree index structure is used with the following extensions. Each B+tree consists of the root element interconnected with the undefinition module (NULL_ORIG_MODULE). Internal nodes

reference traverse path, however, the balancing itself is done outside of the main transaction. The leaf layer does not deal with the ROWIDs pointing to the database repository – file, block, and position, instead, the Multi-index Inter-Stitching layer (MIS layer) is referenced. Thanks to that, several indexes can be evaluated for a separate query. The results of the pre-index processing are shifted into the MIS layer, where the bitmap index is used. The inputs are delimited by the indexes themselves holding the value 1, if the condition is evaluated as TRUE, holding 0 otherwise. Such binary operation applied to the multiple indexes is fast to be applied and processed. The result of the bitmap processing is a list of physical pointers to the database – ROWIDs. Thus, the index itself does not point to the physical repository, it is just part of the MIS module. Thanks to that, if the database repository is moved, there is no necessity to reconstruct the index set, only bitmapper is re-initialized and re-calculated.

Fig. 5 shows the architecture in the index point-of-view (multi-indexing layer). Individual indexes are passed as inputs to the MIS layer, which provides a list of ROWIDs in the output. To improve the performance, the output list can be grouped based on the locations by differentiating physical discs, if multiple storage objects are available.

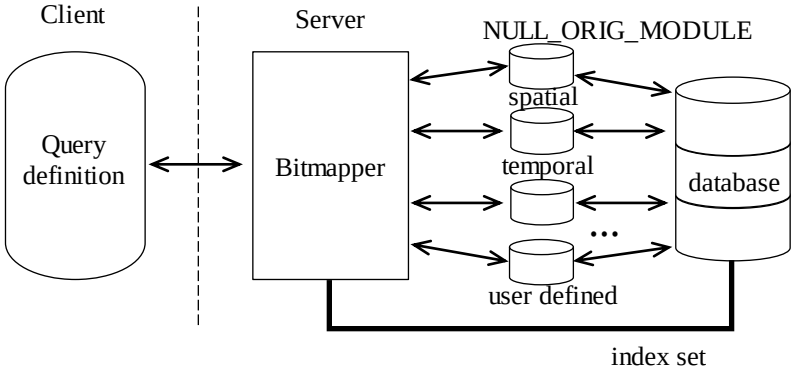

Fig. 5. Bitmapper

4 Performance evaluation

Performance characteristics have been obtained by using the Oracle 19c database system based on the relational platform. For the evaluation, a table containing 10 attributes originated from the sensors was used, delimited by the composite primary key consisting of two attributes. The table contained 1 million rows. The environment consisted of 100 vehicles with the limited communication times

Experiment results were provided using Oracle Database 19c Enterprise Edition Release 19.3.0.0.0 - 64bit Production. Parameters of the used computer are:

- Processor: Intel Xeon E5620; 2,4 GHz (8 cores),
- Operation memory: 48 GB DDR 1333MHz
- Disk storage capacity: 1000 GB (SSD).

An evaluation study has been done in the real transportation environment covering ad-hoc networks. Obtained data requests were based on the relevance to the transport, time elements, and usefulness. Namely, the aim is to get the most valuable data sets based on the affiliation to region, route, and time. We have compared four architectures. Model 0 does not deal with the secondary indexes, only spatio-temporal reference as an index is used. Model 1 is delimited by the static index set for each referenced layer - internal interface (e.g. temporal, spatial, regional, etc.). Model 2 limits the data amount only to the relevant tuples for individual requests. Thus, scanning the whole table is optimal, no additional tuples are present. Internally, it is done by the temporary table for each vehicle. It is evident, that such architecture is not feasible in the real environment, we just want to point to the potential optimal solution to determine the impact of the proposed architecture, as well as additional cost demands. Model 3 covers the proposed multi-indexing solution processed dynamically using the registration. As evident, the proposed multi-indexing model is robust, can cover the reflection of the suitability. Data retrieval is dynamic, processing can be done in massive parallelism, so the processing and data transfer can be done in a fast manner [7] [8].

Fig. 6 shows the results of the processing time. It is expressed in the percentage. By using just the primary key, data must be mostly obtained by using sequential scanning. It reflects 100% of the demands. Internal interface lowers the demands to 40,43%. In that case, the optimizer selects one of the indexes, which is used for the data retrieval using the ROWID. Model 2 provides an optimal solution, the index set is not relevant, whereas the inner table contains just the required data, so the sequential scanning is optimal (in our case, no data fragmentation is present). In that case, processing time demands are 26,43%. In the complex architecture, Model 2 would provide the optimal solution. Our proposed multi-indexing architecture (Model 3) reflects the result of 28,31%. In comparison with Model 2, multiple indexes can be used, followed by the bitmap merging in the second phase. As clear from the fig. 6, the additional demands are only minimal, however, such a solution is robust and sustainable.

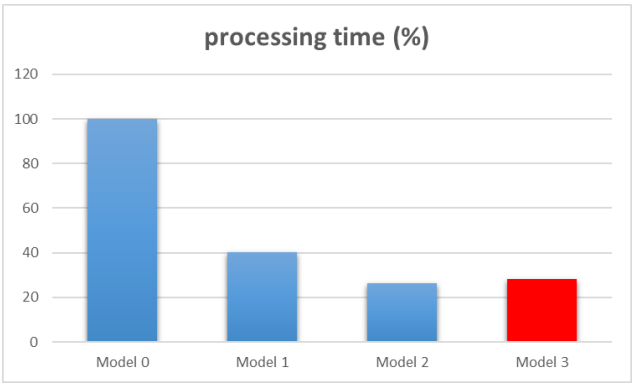

Fig. 6. Processing time demands (%)

Note, that the experiments were executed 10 times reflecting the defined attribute set forming the selection operation. 10 regions were used, data of two ones were required with various importance rates – 100% and 60% for the second [5].

The second evaluation stream is based on the limiting data amount to be transferred. In the first part, the processing time is limited to the value of 1 second. In that case, for Model 2, all required data are obtained. Model 3 was able to transfer only 95,43%, the first region data are processed in the first phase, they are always delivered completely. Model 0 does not use an additional index set, so the available data amount was just 32,12%. Although Model 1 provides all suitable index sets, they cannot be evaluated in parallel, just one of them can be selected, thus, based on the time limitation, just 73,57% of the required data was delivered. Fig. 7 shows the results in a graphical form.

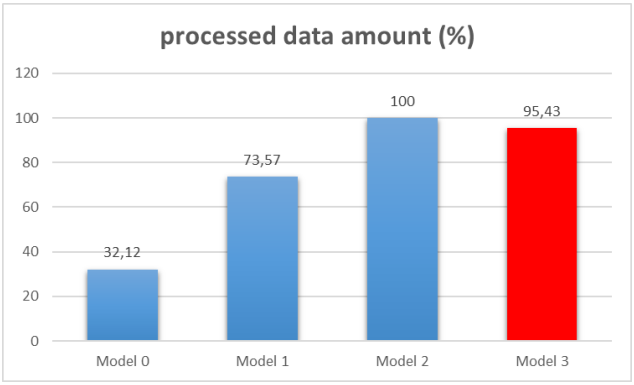

Fig. 7. Processed data amount (%)

We have also experimented with the efficiency of the transfer evaluated as the proportion of the data amount to the total available time. In Model 3, the correlation is almost linear, if the time is limited to the value 500ms, the required data amount proportion is 48,12% when limiting the data to the 250ms, the data amount is approximately 21,89%. Note, if the time availability is lower than 75ms then the data availability is almost zero – fig. 8. The reason is based on the data identification, query definition, and data transfer in the last phase.

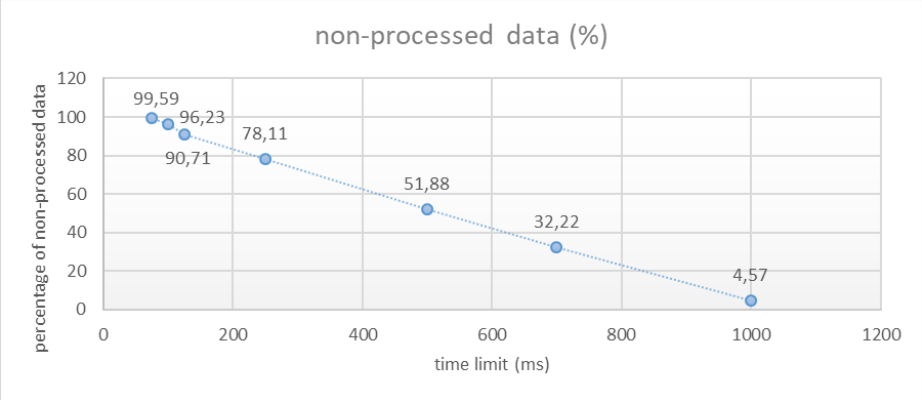

Fig. 8. Non-processed data reflecting the time limit

5 Conclusions

Data efficiency is a complex requirement in any intelligent information system. Data retrieval in an ad-hoc network has to be strictly optimized, whereas the availability time sphere is low. Existing conventional index approaches do not solve the problem. Dynamic index specification using the auto-indexing approach is not suitable, as well, whereas the data and query structure is so dynamic, that the system would mostly point to the index definition and management. Our proposed solution is based on the multi-indexing layer, which can be processed using internal interfaces in parallel. Individual data requests are categorized based on the importance, so the most relevant data are processed in the first phase, whereas the available time for the transfer is limited. The provided solution is based on the B+tree index layer, where individual leaf nodes do not point to the physical data blocks themselves. Instead, they are routed into the second layer, by which the bitmap can be applied. Thanks to that, several indexes can be processed in parallel, thus the processing evaluation can be done more effectively. Data synchronization is present in the bitmap layer by pointing the processing to the physical data blocks to be loaded.

In the future, our emphasis will be on the dynamic block structure size definition based on the data relevance. There is an assumption, that such implementation can provide a better performance, whereas the I/O operation demands are lowered. The dynamic balancing in such a manner would be important (move data across multiple block size levels).

Acknowledgment

This publication was realized with support of the Operational Programme Integrated Infrastructure in frame of the project: Intelligent systems for UAV real-time operation and data processing, code ITMS2014+: 313011V422 and co-financed by the European Regional Development Found.

References

1. Abdalla, H. I.: A synchronized design technique for efficient data distribution, Computers in Human Behavior, Volume 30, 2014, pp. 427-435
2. Burleson, D. K.: Oracle High-Performance SQL Tuning, Oracle Press (2001), ISBN - 9780072190588
3. Jakóbczyk, M.: Practical Oracle Cloud Infrastructure: Infrastructure as a Service, Autonomous Database, Managed Kubernetes, and Serverless, Apress (2020)
4. Jánošíková, Ľ., Kvet, M., Jankovič, P., Gábrišová, L. (2019). An optimization and simulation approach to emergency stations relocation. In Central European Journal of Operations Research, ISSN 1435-246X, Roč.27, č.3 (2019), pp. 737-758.
5. Kvet, M.: Database Index Balancing Strategy, in print (2021)
6. Kvet, M., Kršák, E., Matiaško, K.: Study on effective temporal data retrieval leveraging complex indexed architecture, Applied Sciences 10 (2020)
7. Kvet, M., Matiaško, K.: Analysis of current trends in relational database indexing, conference SST 14-16 October 2020, Osijek, Croatia (2020)
8. Skrinarova, J., Povinsky, M.: Parallel simulation of tasks scheduling and scheduling criteria in high-performance computing systems, Journal of Information and Organizational Sciences (2019)
9. Steingartner, W., Eged, J., Radakovic, D., Novitzka, V.: Some innovations of teaching the course on Data structures and algorithms. In 15th International Scientific Conference on Informatics (2019).
10. https://docs.oracle.com/en/cloud/paas/autonomous-database/adbsa/autonomous-auto-index.html