International Journal of Computer Science and Applications,
©Technomathematics Research Foundation
Vol. 13, No. 2, pp. 106 – 120, 2016

CONTENT BASED IMAGE RETRIEVAL USING MODIFIED SCALABLE DISTRIBUTED TWO-LAYER DATA STRUCTURE

STANISLAW DENIZIAK

Kielce Univerity of Technology, al. Tysiąclecia Państwa Polskiego 7
Kielce, 25-314, Poland
s.deniziak@tu.kielce.pl

TOMASZ MICHNO, ADAM KRECHOWICZ

Kielce Univerity of Technology, al. Tysiąclecia Państwa Polskiego 7
Kielce, 25-314, Poland
t.michno@tu.kielce.pl, a.krechowicz@tu.kielce.pl

Nowadays more and more data is stored in the multimedia databases. Because describing images by keywords most often is not sufficient, performing queries in such databases is very problematic. Moreover the amount of data which has to be stored is increasing every year for example because of the rise of resolution in digital cameras. In this paper a new multimedia database structure designed for Content Based Image Retrieval is presented. It is based on Query by Shape method which decomposes an object into features like shape primitives. Since the database has to store very huge amount of data in order to provide high throughput and scalability we propose to use modified Scalable Distributed Two-layer Data Store.

Keywords: CBIR; SD2DS; multimedia data store.

1. Introduction

The multimedia databases are becoming more and more popular nowadays. There are many applications where they are needed, like social media portals (e.g. Facebook, Instagram, Flickr and Google+) or monitoring systems. Because modern cameras produce high resolution images, the amount of data which has to be stored is very huge. Another problem with multimedia databases is their querying. Traditional approaches, based on textual keywords are not sufficient. More advanced techniques incorporating image content features should be used.

In this paper we propose the idea of a multimedia database structure with ability of Content Based Image Retrieval which is based on our previous work described in [Deniziak and Michno2015], Query by Shape (QS). Query by Shape is a method which is based on decomposing an object into features [Deniziak and Michno2015]. Each feature may consists of shape primitive, a color or a texture. In this paper we only use shape primitives. As a data structure for storage we use a modified Scalable Distributed Two-layer Data Store which is highly scalable, distributed data store [Sapiecha and Lukawski2013]. The modification incorporates adding tree structure, comparing algorithm and returning a set of results to the client.

This paper is organized as follows. The Section II presents the survey of image retrieval algorithms. The Section III contains a short review of NoSQL data stores. The idea of Scalable Distributed Two-layer Data Store is described in the Section IV. The Section V shows the motivation of our research. The idea of our database structure is presented in the Section VI. The conclusion of the research is given in the Section VII.

2. Image Retrieval algorithms

In the area of multimedia databases, three types of retrieval algorithms can be distinguished: Keyword-Based Image Retrieval (KBIR), Content-Based Image Retrieval (CBIR) and Semantic-Based Image Retreival(SBIR).

The first group, KBIR, is based on the relational database approach, where images are described using keywords. During the query, the proper keywords should be given. The database structure is very simple but strongly relies on the textual annotations given by a human. This approach is prone to mistakes because of the subjective kind of descriptions [Deniziak and Michno2015]. Moreover it is very hard to cover the whole information which is present in the image using only textual description [Li and Hsu2008, Wang et al2010].

The CBIR algorithms are based on different approach than KBIR. Image features are used to index images and perform queries [Deniziak and Michno2015]. All algorithms in this group could be divided into two categories: low-level and high-level algorithms. The low-level algorithms process images globally, extracting features from the whole frame, using e.g. a normalized color histogram [Mocofan et al2011], a spatial domain [Shih2002], a difference moment and entropy [Kriegel et al2006] or an MPEG-7 image descriptors like shape and texture [Lalos et al2008]. The low-level features used by CBIR algorithms are easy to compute but they are insufficient if the query is oriented on searching for similar objects rather than whole images, which incorporates separating the object from the background.

The high-level CBIR algorithms provide more reliable and precise results in this situation. The major part of the algorithms from this group are based on the regions extraction and graphs matching. A region is a group of similar pixels, most often grouped by colors. There are also methods which uses during region extraction: a set of primitives [Jakubowski1985] or fuzzy pattern detection [Bielecka and Skomorowski2007], moment-based local operators [Sluzek2005] or parallelograms, ellipses, corners and arcs detection [Lee and Fu1983]. After region extraction, a graph is being constructed in order to store the relations between them [Deniziak and Michno2015]. During the multimedia database query, the graph-subgraph matching is performed e.g. using the classic Ullman algorithm [Ullmann1976] or more advanced ones. There are algorithms that are automatic or semi-automatic with ability to present preliminary results to the user who may choose important regions and repeat the query [Aggarwal et al2002]. Another group of CBIR algorithms allows queries without full knowledge about searched images or objects. One of the first and most successive approaches in this area uses a human drawn sketches which are compared with the corresponding sketches in the database, globally for the

whole image [Kato et al1992]. Another example is Query by Shape method [Deniziak and Michno2015], which is our previous research and which is used as a base of this paper. The idea of the algorithm is to decompose objects into features like shape primitives or color features. The features are not used for region extraction but for constructing an object's sketch or skeleton which is strictly a graph. Also the matching algorithm is proposed which uses the similarity coefficient. The similarity informs how similar two graphs are. If they are the same, the similarity is equal to 1, if they are completely different, it takes 0 value. All intermediate values indicates that graphs are partially similar.

The SBIR algorithms are algorithms that try to overcome the "semantic gap", which is the difference between what is present on the image and what a human could interpret [Wang et al2010, Singh et al2012]. Most SBIR algorithms are based on textual description which, in contrast to the KBIR algorithms, is a much longer phrase. The phrase is easy to create, use and understand by the human, the example could be "the sunrise in the mountains". The textual descriptions are not used directly, but they are mapped onto semantic features and then a query is performed [Li and Hsu2008].

All groups of algorithms need efficient data storage methods. One of the most often used structure is a cell or a tree, because it can store the relations between similar images [Lalos et al2008]. There are also approaches that joins both data structures. One of the example is [Kiranyaz and Gabbouj2007] which is based on gathering similar records in the same cells. Additionally, some biological processes are added, like mitosis, when the similarity between items in the same cells is below the specified level.

The research described in this paper is devoted to CBIR algorithms due to the fact that they are able to provide precise results for queries performed both by a human and by an automatic image detection which is the aim of our main research.

3. The NoSQL data stores

The multimedia databases very often stores millions of photos or images. Because there has to be stored and processed very huge amount of data, with high availability and workload management, the traditional SQL-based databases may not be sufficient. Much better results are obtained using NoSQL databases. The NoSQL databases may be classified into the following groups: Graph databases, Key-Value data stores, Document data stores and Column-based data stores [Sadalage and Fowler2013].

The Graph databases provides similar features as relational databases but they are not well-prepared to store huge amount of data because of the problems with scalability [Sadalage and Fowler2013]. Key-Value data stores use a unique single key (e.g. a number) for storing data (value). Most often they have only two operations: inserting and retrieving data [DeCandia et al2007]. Even the operation of updating and deleting are missing in many cases. In others, they exists but are not very effective. The examples of such data stores are Dynamo data store by Amazon which uses single integer key [DeCandia et al2007] or Apache Hbase which identifies the data by a timestamp, column name and row name [Chang et al2008]. The Document data stores use textual data

interchange formats, like JSON or xml to store data. The most popular document data store is MongoDB [Sadalage and Fowler2013]. There is still no standard in case of the NoSQL systems. There is a great diversity in this world. Almost every system is created for special purpose and can be suitable in different application.
The NoSQL data stores were successfully adapted to store images. As an example the Apache Hbase may be given, which was used to store Google Earth images [Chang et al2008]. However, most of them are still focused on storing data portions of relatively small size. For example MongoBD allows to store documents up to 16MiB [MongoDB2015]. In case of MongoDB there is a special API, called GridFS, that allows to store bigger objects. Unfortunately, storing data in GridFS have serious limitations. The data can not be queried like the smaller ones and can not be processed in the store. Still in the vast majority of the NoSQL systems the practice rule is to store only the metadata while the actual data should be stored in raw format on the file systems. However, this solution is the most popular and effective it does not allow advanced data processing in the store.

4. The SD2DS data stores

Scalable Distributed Two-layer Data Structures (SD2DS) are one of variants of Scalable Distributed Data Structures (SDDS) [Sapiecha and Lukawski2013], which may be classified as a distributed NoSQL data store. The SDDS is a very mature standard but due to the growing interests of the Cloud systems it becomes more and more useful [Wang et al2015]. The SD2DS structure is presented in the Fig. 1. The SD2DS store data using a key and a value in so-called “buckets”. Each bucket has storage limit and when it is reached a split is performed. Buckets may be distributed through many machines, e.g. nodes on the multicomputer. The SD2DS stores data in two separate layers. The first layer contains only data headers which consists of the data locator and metadata. The data locator is used to locate the stored data in the second layer. The metadata is an additional information connected with the actual data and may contain optional information e.g. the data length, a checksum, insertion date or data description. The second layer contains only the actual data, called body [Sapiecha and Lukawski 2013]. The body and the corresponding header creates so called component. Every clients operation on the structure is composed with two queries. At first client query the first layer using the key. He obtain the header with the locator. By using this locator client can query actual data in the second layer. That structure maintains big portions of data more efficiently, because both layers are independent of each others and may be stored on different machines. Moreover the data store is highly scalable, without theoretical limitations and may contain as many data as is needed. The SD2DS allows also to use as a second layer other solutions, even without buckets.

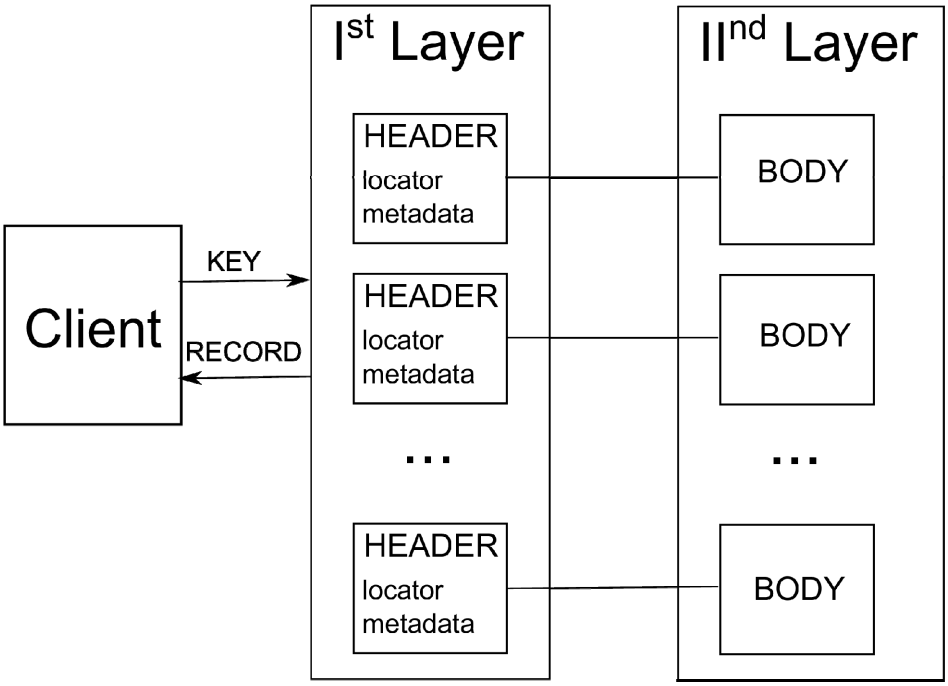

Figure 1: The SD2DS structure overview.

The SD2DS may be easily extended e.g. by adding throughput adjustment [Sapiecha et al (2014)] which highly improves the data access time. The vast majority of that kind of extensions are simply introduced by adding new metadata to the first layer of the system. This feature gives us the opportunity to prepare the data store for the specialized custom needs. Moreover, because of the independence of the two layer the whole system can be easily located in the Cloud. It can even be used to integrate different infrastructure fitted to the actual needs.

Recently, we developed fast and efficient storage on the base of SD2DS architecture [Krechowicz et al2015]. The conception of SD2DS proven to be very effective in designing such kind of systems. We were able to seriously compete with well known systems that are openly available. This practical implementation gave us opportunity to evaluate the proposed image retrieval methods in real-world environment. Our data store is specially useful for storing relatively big components. This feature is especially important because the resolution of images is growing all the time. The example evaluation of our storage in case of the big components (10MiB) is presented on Fig. 2. We evaluate the processing time of the data in relation to the number of clients that simultaneously operates on the systems. As it can be seen our storage was able to outperform well known NoSQL representative MongoDB. Even in case of the Memcached, which is very effective solution for memory cache, our system can be treated as a serious competitor.

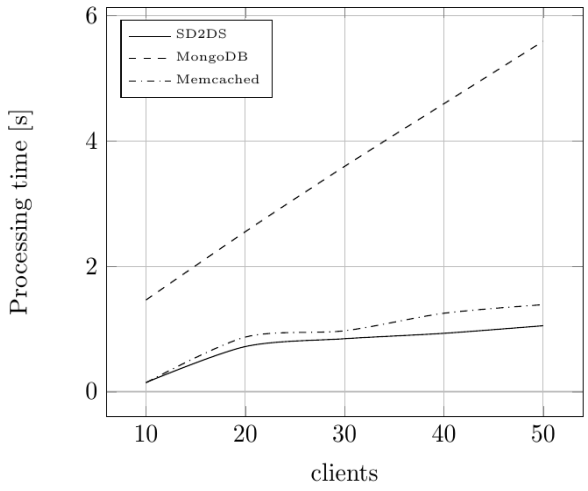

Figure 2: Time comparison of getting components of fixed (10MiB) size [Krechowicz et al2015]

5. Motivation

The problem of multimedia database querying is very complicated. As a continuation of the previous research [Deniziak and Michno (2015)], we would like to propose the Content Based Image Retrieval Database incorporating Query by Shape Method. The database has to store a very huge amount of data. Simple records table, a tree or a cell would be not sufficient. Because of that, more advanced data stores, especially NoSQL-based should be used. In order to obtain very high scalability and availability, as a base for our system the Scalable Distributed Two-Layer Data Store should be used. As a result of our research we would like to create the system which will fulfil the following requirements:

- queries using a graph of features e.g. primitives,
- similarity control for graphs comparison, the similarity should be set by the client,
- client feature: graphical queries easily drawn by a human (without drawing skills), e.g. using predefined shapes,
- client feature: automatic image transformation into a graph, used for a query,
- very fast data access,
- graph comparisons on the server side, in order to allow usage of low performance client devices
- scalability without turning off the system,
- easy addition of a new hardware used for storage,
- good performance for very huge data workload stored in the database.

6. The system overview

The Query by Shape image retrieval data store is based on a modified SD2DS data store structure [Deniziak et al2015]. The SD2DS data store consists of the record's headers (the first layer) and their bodies (the second layer). In order to retrieve the record, its key in the SD2DS have to be provided. Another operations allowed are updating and deleting the record. Because Query by Shape algorithm uses graphs, in order to provide fast access, we insert them into a tree. Figure 3 shows the modified SD2DS data store in order to provide such a feature. Since that tree may contain many records, we propose to store all nodes in SD2DS first layer header.

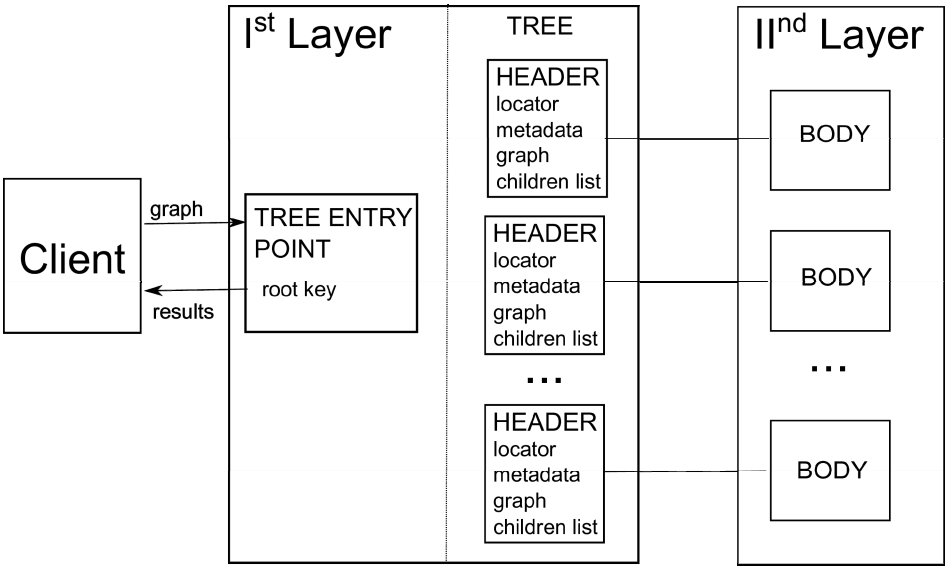

Figure 3: The proposed system overview.

6.1. *The Client*

One of the main differences between our proposition and classical SD2DS is the communication of the client with the store. During the query, the client should send a graph which will be used during comparisons with records in the database. Moreover it has to be able to receive the result of a query which may consists of many records. Therefore, the following messages are defined [Deniziak et al2015]:
Messages send by the client:

- record (graph and image) insertion (GI) - a message used to insert a new record containing a graph and an image into the database,
- query (SQ) - a query after which all result records are sent by the database in a one message,

- distributed query (DQ) -a query after which result records are sent gradually with one message per one record, allowing e.g. to show progressively results for a user,
- get record (GR) - a message used to retrieve the record using its key.

Messages received by the client:

- result of the insertion (RI) - sent by the database as a response to GI message, returning the failure code or new node's record key,
- not found (NF) - sent by the data store when there are no similar graphs/images in the data store, returned for both SQ and DQ messages,
- results (RR) - contains list of similar records, returned only as a response to SQ message,
- single result (SR) - contains only one similar record, returned only as a response to DQ or GR messages.

All sent messages contain the query graph and the minimal similarity which is used during comparisons. Moreover the messages may define if as a result full record or only a header should be returned.

6.2. *The First Layer*

The first layer of the data store was redesigned into two sub-layers: the Tree Entry Point and the Tree. The Tree sub-layer consists of the standard SD2DS record headers. Each record header is treated as a one tree node, containing as a metadata a graph and a list of children nodes (a list of records keys). Moreover, each bucket is able to execute query and traverse the tree on its stored nodes and to communicate with other buckets in order to finish that operations. The Tree Entry Point is responsible for communicating with the clients and the logic of the tree: storing the tree root record key, inserting new nodes into the tree structure, modifying nodes, managing the query and traversing the tree. Moreover it is able to execute queries in two ways, analogously to the SQ and DQ messages. In order to improve the performance and decrease the probability of failures, there may be more than one Tree Entry Point. For example when there are many clients, each of them may be connected to the different Tree Entry Point, balancing the workload.

6.3. *The tree structure*

The tree nodes used in the database are stored in the first layer using standard SD2DS headers records. The graphs are inserted into the tree hierarchically, storing its common parts in parent nodes. Because of that, the root node may contain a graph with only one shape or an empty graph. The example tree structure with scooter, bicycle and car objects is shown in the Fig. 4. Moreover, because there are indirect nodes which stores only graphs without connection to any images, the SD2DS modification allows records headers in the first layer with empty bodies.

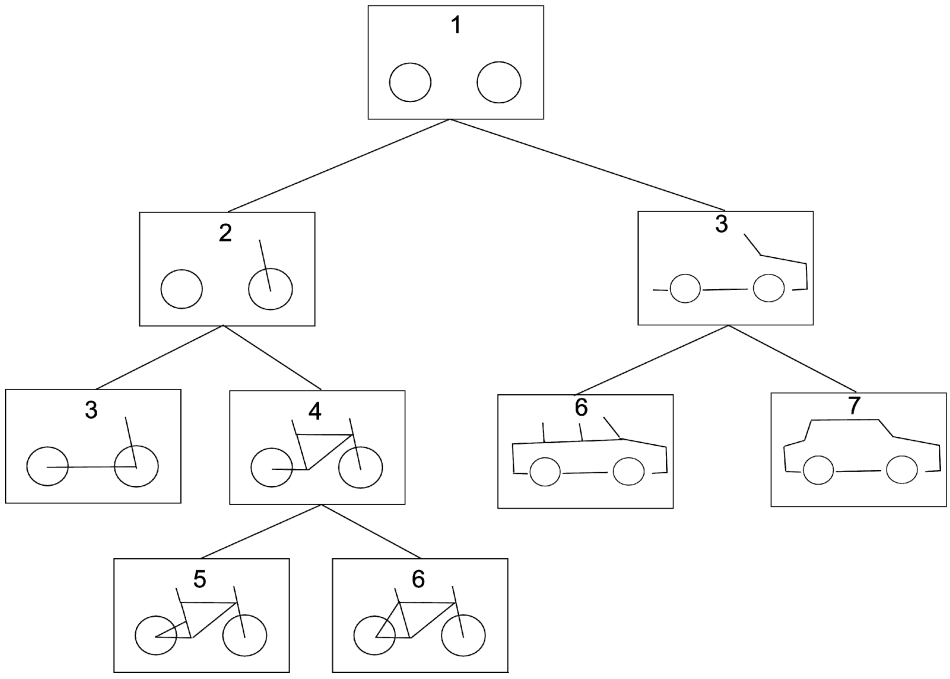

Figure 4: The example tree with following objects stored as a nodes: bicycles (4, 5, 6), scooter (3) and cars (6, 7). There are also shown intermediate nodes: common for scooter and bicycles (2), for cars (3) and for all object classes (1, the root key).

Record Insertion

After receiving the GI message the Tree Entry Point compares the graph with graphs stored in the tree, starting from the root node using the Query by Shape comparison algorithm (QS) [Deniziak and Michno2015] and the minimal similarity parameter extracted from the GI message. The algorithm is as follows:

(1) If there are no nodes in the tree:
  (i) add new node as a root node, store its key as a Root Key
  (ii) send Root Key as a RI message to the client
  (iii) exit algorithm
(2) add Root Key to the FIFO queue
(3) while there are still elements in the FIFO queue:
  (i) pop the first element from the FIFO
  (ii) get record dbRecord with the key equal to the popped key element.
  (iii) compare graph with dbRecord graph using QS algorithm, store the similarity sim result
  (iv) If dbRecord is not a root node:
    If sim ≤ of similarity with dbRecord's parent:
    i. add graph as a child of dbRecord's parent
    ii. send graph key to to client as a RI message
    else

iii. add graph as a new root node
iv. add dbRecord to the graph's children list
v. send graph key to to client as a RI message

(v) If sim ≥ minimal nodes similarity:
i. add record as a child to dbRecord;
ii. send graph key to to client as a RI message
else
iii. If dbRecord does not have any children:
- add graph to the tree
- add the common part of the graph and dbRecord as their parent, if dbRecord is a root key, update
- send graph key to to client as a RI message

else
- add all dbRecord children to the FIFO queue.

The Tree Entry Point retrieves the record using similar procedures as the client in the standard SD2DS. Then children nodes are extracted from the record and used for the next records retrieval.

Querying

The querying algorithm is very similar to the insertion of a node. During the query, the results could be sent immediately to the client or stored in the Tree Entry Point temporary buffer before completion and then sent. The algorithm is as follows:

(1) If tree is empty send NF message to the client.
(2) Add the root key to the FIFO queue.
(3) While there are elements in the FIFO:
(i) pop the first element from the FIFO
(ii) get record dbRecord with the key equal to the popped key element
(iii) compare graph with dbRecord graph using QS algorithm, store the similarity sim result
(iv) if sim ≥ highSimilarity:
i. if the client client sent DQ message:
- send RI message with the record's key to the client
- send RI message with all keys of the sub-tree consisting of record's children to the client

else
- add record to the results
- add all keys of the sub-tree consisting of record's children to the client

(v) if similarity with dbRecord's parent − minSimilarity > sim then continue
(vi) if dbRecord has children then add each of them to the FIFO queue

6.4. *The Second Layer*

The second layer has not been modified. Because of the tree structure, there may be less bodies than records in the first layer. The images are retrieved during querying, but the client in the SQ or DQ could force the database to only send headers from the first layer or even only the keys.

7. Experimental results

The architecture described in following chapters has been tested using a set of applications written in C++. In order to evaluate the performance, the tree version stored in the MySQL database was also implemented.
The first experiments included tests for evaluation of how many correct images are returned by the system and how many of them were still not retrieved. In order to compare the results, the commonly used *precision* and *recall* coefficients were used:

$$precision = \frac{number\ of\ relevant\ results\ images}{total\ number\ of\ results\ images} \quad (1)$$

$$recall = \frac{number\ of\ relevant\ results\ images}{total\ number\ of\ relevant\ database\ images} \quad (2)$$

The test results for the two example bicycle and cars images are shown in the Table 1. As the data set, 117 real life images of different object classes were used. The manual queries were performed using the prototype GUI which allowed to draw predefined shapes (lines and circles) which were then transformed into an object graph. The automated queries were based on automatic shape (also line and circles) detection using Linear and Circular Hough Transform algorithms. After detection, similarly as in manual queries, all shapes were transformed into a graph. The lower precision and recall results of automated queries are caused by some inaccuracies during the image decomposition stage (which consisted of shape detection). Moreover, they contain many unconnected nodes in graphs which were problematic for inserting them into the tree. In order to achieve higher recall and precision values, more advanced shape detection algorithms have to be used. Moreover some preprocessing after detection may be applied, e.g. merging splitted parts of lines.

Table 1: The precision and recall results for the example bicycle and car queries

	Manual queries		**Automated queries**	
	Precision	**Recall**	**Precision**	**Recall**
bicycle no. 1	0.7708	0.9737	1	0.3023
bicycle no. 2	0.8529	0.7636	1	0.2558
car no. 1	1	0.5814	0.4182	0.5
car no. 2	1	0.7209	0.5902	0.72

Another set of experiments were prepared in order to test the performance of the system. The usage of the SDDS was evaluated comparing the results with the MySQL storage tree implementation. Two different clients were used, no. 1 with the same CPU frequency as server and no. 2 with CPU frequency 60% lower than server. For the MySQL implementation all graph comparisons were performed on the client, for SDDS they were performed in the buckets on the server. The results are presented in the Table 2, Fig. 5 and Fig. 6. It can be seen that for the client no. 1 the times are very similar for both SDDS and MySQL. The faster SDDS data access time is overcome by the long graphs comparison time. For the client no. 2 the SDDS implementation is much faster than MySQL. This is mainly caused by the 60% lower client CPU frequency then in client no. 1. For the MySQL implementation each node has to be retrieved to the client and then compared with the query graph. For the SDDS implementation the client performance does not have big influence on the query time because all comparisons are performed in the buckets.

Table 2: The query time for different number of images (in milliseconds).

		Total number of images				
client	storage	10	50	100	200	500
no. 1	SD2DS	869	2572	4329	8229	20733
	MySQL	857	2942	5384	11248	21524
no. 2	SD2DS	819	2466	4356	8185	20430
	MySQL	1461	4964	9681	17662	44045

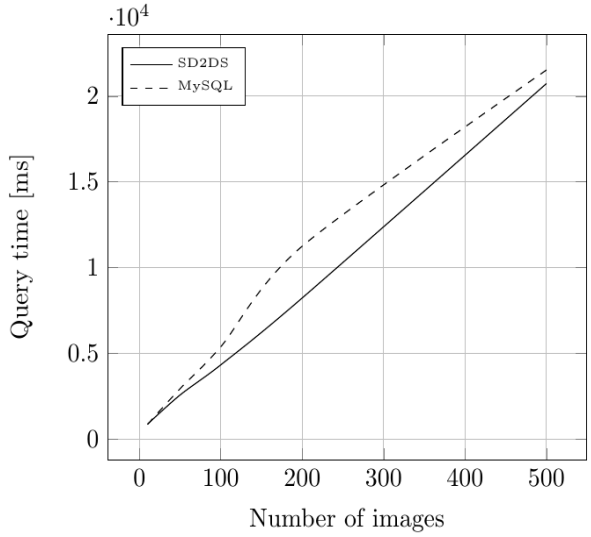

Figure 5: The query time for different number of images (in milliseconds) for the Client no. 1.

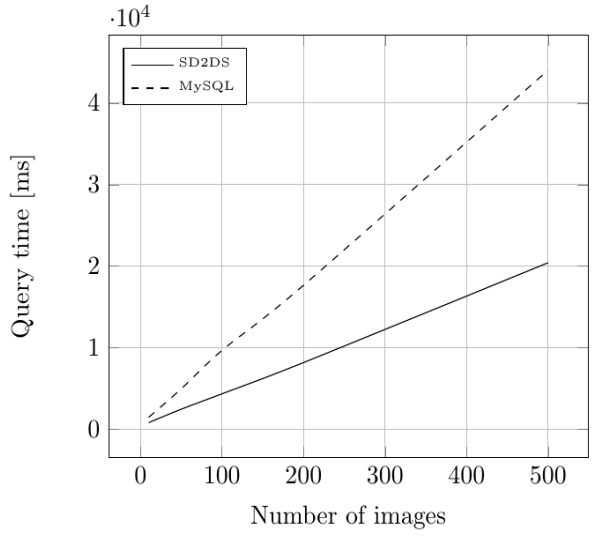

Figure 6: The query time for different number of images (in milliseconds) for the Client no. 2.

8. Conclusion

In this paper a new Content Based Image Retrieval database structure was presented. It incorporates our previous research, the Query by Shape method [Deniziak and Michno2015] which is based on object decomposition into shapes and the idea of SD2DS database. As a structure for storing graphs we propose a tree which nodes are placed in the SD2DS buckets. Therefore, the SD2DS structure has to be modified, e.g. the first

layer has to be redesigned and a Tree Entry point has to be added. The first test results are very promising.

As the future research we concern more advanced Query by Shape tree integration with SD2DS database, e.g. improved communication between nodes in buckets. Moreover, some throughput adjustment has to be implemented into the structure in order to allow better client-server communication performance.

Another direction of research will concern the Query by Shape method in order to improve the results precision. This may include e.g. adding other primitive types and color features. Moreover more advanced shape extraction and graph matching algorithms should be applied, as well as some optimization methods should be evaluated, e.g. [Sitek and Wikarek2015].

Acknowledgment

The research used equipment funded by the European Union in the Innovative Economy Programme, MOLAB - Kielce University of Technology.

References

Aggarwal, G., Ashwin, T.V., Ghosal, S. (2002). An image retrieval system with automatic query modification. *IEEE Transactions on Multimedia*, **4(2)**: pp. 201–214.

Bielecka, M., Skomorowski, M. (2007). Fuzzy-aided parsing for pattern recognition. *Computer Recognition Systems 2, Advances in Soft Computing*, **vol. 45**: pp. 313–318.

Chang, F., Dean, J., Ghemawat, S., Hsieh, W. C., Wallach, D. A., Burrows, M., Chandra, T., Fikes, A., Gruber, R. E. (2008) Bigtable: A distributed storage system for structured data. *ACM Trans. Comput. Syst.*, **26(2)**: pp. 1–26.

DeCandia, G. Hastorun, D., Jampani, M., Kakulapati, G., Lakshman, A., Pilchin, A., Sivasubramanian, S., Vosshall, P., Vogels, W. (2007) Dynamo: Amazon's highly available key-value store. *SIGOPS Oper. Syst. Rev.*, **41(6)**: pp. 205–220.

Deniziak, S., Michno, T. (2015) Query by shape for image retrieval from multimedia databases. *Beyond Databases, Architectures and Structures*, **volume 521** of *Communications in Computer and Information Science*: pp. 377–386.

Deniziak, S.; Michno, T.; Krechowicz, A., (2015) The scalable distributed two-layer Content Based Image Retrieval data store *Computer Science and Information Systems (FedCSIS), 2015 Federated Conference on* pp. 827–832

Jakubowski, R. (1985) Extraction of shape features for syntactic recognition of mechanical parts. *IEEE Trans. on Systems, Man and Cybernetics*, **SMC-15(5)**: pp. 642–651.

Kato, T., Kurita, T., Otsu, N., Hirata, K. (1992) A sketch retrieval method for full color image database-query by visual example. *11th IAPR International Conference on Pattern Recognition, Vol.I. Conference A: Computer Vision and Applications* pp. 530–533.

Kiranyaz, S., Gabbouj, M. (2007) Hierarchical cellular tree: An efficient indexing scheme for content-based retrieval on multimedia databases. *Multimedia, IEEE Transactions on*, **9(1)**: pp. 102–119.

Kriegel, H.-P., Kroger, P., Kunath, P., Pryakhin. A. (2006) Effective similarity search in multimedia databases using multiple representations. *12th International Multi-Media Modelling Conference Proceedings*

Lalos, C., Doulamis, A., Konstanteli, K., Dellias, P., Varvarigou, T. (2008) An innovative content-based indexing technique with linear response suitable for pervasive environments. *International Workshop on Content-Based Multimedia Indexing* pp. 462–469.

Lee, H. Ch., Fu, K. S. (1983) Generating object descriptions for model retrieval. *IEEE Trans. on Pattern Analysis and Machine Intelligence* **PAMI-5(5)**: pp. 462–471.

Li, Ch. Y., Hsu, Ch. T. (2008) Image retrieval with relevance feedback based on graph-theoretic region correspondence estimation. *IEEE Transactions on Multimedia*, **10(3)**: pp. 447–456.

Mocofan, M., Ermalai, I., Bucos, M., Onita, M., Dragulescu, B. (2011) Supervised tree content based search algorithm for multimedia image databases. *6th IEEE International Symposium on Applied Computational Intelligence and Informatics* pp. 469–472.

Sadalage, P. J., Fowler, M. (2013) *NoSQL distilled : a brief guide to the emerging world of polyglot persistence*. Addison-Wesley, Upper Saddle River, NJ, 2013.

Sapiecha, K., Lukawski, G., Krechowicz, A. (2014) Enhancing throughput of scalable distributed two – layer data structures. *Parallel and Distributed Computing (ISPDC), 2014 IEEE 13th International Symposium on* pp. 103–110.

Sapiecha, K., Lukawski, G. (2013) Scalable distributed two-layer data structures (sd2ds) *IJDST*, **4(2)**: pp. 15–30.

Shih, T. K. (2002) Distributed multimedia databases. *chapter Distributed Multimedia Databases* 2–12. IGI Global, Hershey, PA, USA, 2002.

Singh, A., Shekhar, S., Jalal, A.S (2012). Semantic based image retrieval using multi-agent model by searching and filtering replicated web images. *Information and Communication Technologies (WICT), 2012 World Congress on*, pp. 817–821.

Sitek, P., Wikarek, J. (2015) A hybrid framework for the modelling and optimisation of decision problems in sustainable supply chain management. *International Journal of Production Research*.

Sluzek, A. (2005) On moment-based local operators for detecting image patterns. *Image and Vision Computing*, **23(3)**: pp. 287 – 298.

Ullmann, J. R. (1976) An algorithm for subgraph isomorphism. *J. ACM*, **23(1)**: pp. 31–42.

Wang, H.H., Mohamad, Dz., Ismail, N. A. (2010) Approaches, challenges and future direction of image retrieval. *CoRR*.

The MongoDB 3.0 Manual `http://docs.mongodb.org/manual/` *[Online: accessed 14–April–2015]*.

Wang, G., Zomaya, A., Perez, G. M., Li, K. (2015) A Dependable, Scalable, Distributed, Virtual Data Structure *Algorithms and Architectures for Parallel Processing* pp. pp. 724–735.

Krechowicz, A., Chrobot, A., Deniziak, S., łukawski, G. (2015) SD2DS-based Datastore for Large Files *accepted for publication*.