MESSIF: Metric Similarity Search Implementation Framework*

Michal Batko David Novak Pavel Zezula
{xbatko,xnovak8,zezula}@fi.muni.cz
Masaryk University, Brno, Czech Republic

Abstract

The similarity search has become a fundamental computational task in many applications. One of the mathematical models of the similarity – the metric space – has drawn attention of many researchers resulting in several sophisticated metric-indexing techniques. An important part of a research in this area is typically a prototype implementation and subsequent experimental evaluation of the proposed data structure. This paper describes an implementation framework called MESSIF that eases the task of building such prototypes. It provides a number of modules from basic storage management to automatic collecting of performance statistics. Due to its open and modular design it is also easy to implement additional modules if necessary. The MESSIF also offers several ready-to-use generic clients that allow to control and test the index structures and also measure its performance.

1 Introduction

The mass usage of computer technology in a wide spectrum of human activities brings the need of effective searching of many novel data types. The traditional strict attribute-based search paradigm is not suitable for some of these types since they exhibit complex relationships and cannot be meaningfully classified and sorted according to simple attributes. A more suitable search model to be used in this case is *similarity searching* which can be based directly on the data content rather than on extracted attributes.

This research topic has been recently addressed using various approaches. Some similarity search techniques are tailored to a specific data type and application, others are based on general data models and are applicable to a variety of data types. The *metric space* is a very general model of similarity which seems to be suitable for various data and which is the only model available for a number of important data types – e.g. in multimedia processing. This concept treats the dataset as unstructured objects together with a *distance* (or *dissimilarity*) measure computable for every pair of objects.

Number of researchers has recently focused on indexing and searching using the metric space model of data. The effort resulted in general indexing principles defining fundamental main-memory structures, continued with designs of disk-based structures and also with proposal of distributed data-structures which enable efficient management of very large data collections.

An important part of research in this area is typically a prototype implementation and subsequent experimental evaluation of the proposed data structure. Individual structures are often based on very similar underlying principles or even exploit some existing structures on lower levels. Therefore, the implementation calls for a uniform development platform that would support a straightforward reusability of code. Such a framework would also simplify the experimental evaluation, make the comparison more fair and thus the results would be of greater value.

This reasoning led us to a development of MESSIF – The Metric Similarity Search Implementation Framework. Its design pursue the following objectives:

- to provide *basic support* for the indexing based on metric space – let developers focus on the higher-level design;
- to create a unified and semi-automated mechanism for *measuring* and collecting *statistics*;
- to define and use uniform interfaces to support *modularity* and thus allow *reusing of the code*;
- to provide infrastructure for *distributed processing* with focus on peer-to-peer paradigm – communication support, deployment, monitoring, testing, etc.;

*This research has been funded by the following projects: Network of Excellence on Digital Libraries (DELOS), national research project 1ET100300419, and Czech Science Foundation grant No. 102/05/H050.

- to support complex similarity search in *multi-metric* spaces.

The rest of the paper maps all components of the MESSIF platform from basic management of metric data in Sections 2 and 3, over the support for the distributed processing in Sections 4 and 5 to the multi-metric support in Section 6 and user interfaces in Section 7. The text is subdivided into shorter parts each of which describes one logical component of MESSIF – its theoretical background, specific assignment for the framework (MESSIF Specification) and description of currently available modules that provide the required functionality (MESSIF Modules). The architecture of the framework is completely open – other modules can be integrated into the system in a straightforward way.

2 Metric Space

The metric space is defined as a pair $\mathcal{M} = (\mathcal{D}, d)$, where $\mathcal{D}$ is the domain of objects and d is the total distance function $d : \mathcal{D} \times \mathcal{D} \longrightarrow \mathbb{R}$ satisfying the following conditions for all objects $x, y, z \in \mathcal{D}$:

$$\begin{array}{ll} d(x,y) \geq 0 & (\textit{non-negativity}), \\ d(x,y) = 0 \text{ iff } x = y & (\textit{identity}), \\ d(x,y) = d(y,x) & (\textit{symmetry}), \\ d(x,z) \leq d(x,y) + d(y,z) & (\textit{triangle inequality}). \end{array}$$

No additional information about the objects' internal structure or properties are required. For any algorithm, the function d is a black-box that simply measures the (dis)similarity of any two objects and the algorithm can rely only on the four metric postulates above.

MESSIF Specification

Our implementation framework is designed to work with a generic metric space objects. The internal structure of the objects is hidden and not used in any way except for the purposes of evaluation of the metric function. In particular, every class of objects contains an implementation of the metric function applicable to the class data.

For the purposes of quick addressing, every object is automatically assigned a unique identifier *OID*. Since the metric objects are sometimes only simplified representations of real objects (e.g. a color histogram of an image), the objects also contain a URI locator address pointing to the original object – a web address of an image for instance.

MESSIF Modules

- **Vectors** – numeric vectors of an arbitrary fixed dimension. Implemented vector metric functions: L_p metric for any p, L_{inf} metric, quadratic form distance with a given matrix, Earth Mover's Distance [17].
- **Strings** – variable length strings. Implemented metric functions: edit distance, weighted edit distance with a specified cost matrix, protein distance function.

2.1 Collections and Queries

Let us have a collection of objects $\mathcal{X} \subseteq \mathcal{D}$ that form the database. This collection is dynamic – it can grow as new objects $o \in \mathcal{D}$ are inserted and it can shrink by deletions. Our task is to evaluate queries over such a database, i.e. select objects from the collection that meet some specified similarity criteria. There are several types of similarity queries, but the two basic ones are the range query $\mathbf{Range}(q, r)$ and the k-nearest neighbors query $\mathbf{kNN}(q, k)$.

Given an object $q \in \mathcal{D}$ and a maximal search radius r, *range query* $\mathbf{Range}(q, r)$ selects a set $S_A \subseteq \mathcal{X}$ of indexed objects: $S_A = \{x \in \mathcal{X} \mid d(q, x) \leq r\}$.

Given an object $q \in \mathcal{D}$ and an integer $k \geq 1$, *k-nearest neighbors query* $\mathbf{kNN}(q, k)$ retrieves a set $S_A \subseteq \mathcal{X} : |S_A| = k, \forall x \in S_A, \forall y \in \mathcal{X} \setminus S_A : d(q, x) \leq d(q, y)$.

MESSIF Specification

In MESSIF, we introduce concept of *operations* to encapsulate manipulations with a collection. An operation can either modify the collection – insert or delete objects – or retrieve particular objects from it. Every operation carries the necessary information for its execution (e.g. an object to be inserted) and after its successful evaluation on the collection it provides the results (e.g. a list of objects matching a range query). If the operation is a query, it also provides an implementation of its basic evaluation algorithm – the *sequential scan*. It is a straightforward application of the particular query definition: given a collection of objects, the operation inspect them one by one updating the result according to that particular query instance.

MESSIF Modules

- **Insert operation** – used to insert an object into the database. The object being inserted is the only argument of the operation.
- **Delete operation** – allows removal of an object from the database. The identifier OID of the object to be deleted is provided.
- **Range query operation** – carries the query object q and the radius r. The sequential scan implementation evaluates $d(q, o)$ for every object o from the given set adding o to the result if the distance is up to r. This distance is also used for the object relevance ranking.
- **kNN query operation** – contains the query object q and the number of nearest neighbors k. The sequential scan implementation evaluates $d(q, o)$ for every object o from the given set maintaining the sorted result list of k objects with smallest distances.
- **Incremental kNN query operation** – the query object q is specified. Every time the operation is executed, additional neighbors (a specified number of them) are added to the result. The sequential scan is implemented exactly as the kNN query only skipping the objects already in the result set.

3 Metric Data Management

We have explained the concept of the metric-based similarity search. In this section, we will focus on efficient management and searching of metric data collections. So far, we can use the aforementioned framework modules to design a primitive data structure – it would execute the sequential scan implementation of a query on the whole collection of generic metric space objects. This works for small and static data sets, but when the data is dynamic and its volume can grow, more sophisticated effectiveness-aimed structures are needed. The framework offers additional modules to simplify the task of implementing such structures – namely the data management support, reference objects choosing (including partitioning) and the encapsulation envelope for algorithms that provides support for operation execution.

A vital part of every implementation is its performance assessment. Without any additional effort required, the framework automatically gathers many statistic values from the summarizing information about the whole structure to the details about local operation execution. In addition, every structure can define its own statistics, which can take advantage of other framework modules.

3.1 Storing the Collections

Above, we have defined the collection as the finite subset of the metric domain $\mathcal{X} \subseteq \mathcal{D}$. Practically, the collection is any list of objects of arbitrary length, which is stored somewhere. For example, the result of any query is a collection too. Moreover, a union of two collections is also a collection and also its subset is a collection.

MESSIF Specification

The collections of objects can be stored in data areas called *buckets*. A bucket represents a metric space partition or it is used just as a generic object storage. The bucket provides methods for inserting one or more objects, deleting them, retrieving all objects or just a particular one (providing its *OID*). It also has a method for evaluating queries, which pushes all objects from the bucket to the sequential scan implementation of the respective query. Every bucket is also automatically assigned a unique identifier *BID* used for addressing the bucket. An example of a bucket is shown in Figure 1b. The buckets have

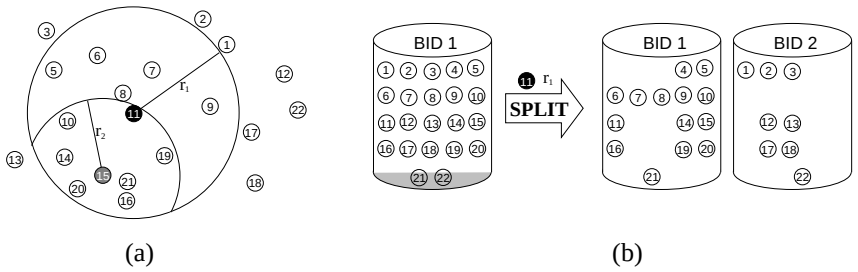

Figure 1: Ball partitioning (a) and a bucket split (b).

usually limited capacity and MESSIF offers methods for splitting them if they overflow as depicted by the figure.

MESSIF Modules

- **Main memory bucket** – bucket is implemented as a linked list of objects in main memory. Objects are inserted at the end of the list and deletions, getting and searching are sequential.
- **Disk storage bucket** – implementation that stores objects persistently on a disk. Block organization is used together with a directory cashed in main memory. Insert and get operations require only one block access, the other operations are sequential with pre-fetching.

3.2 Partitioning the Collections

As the data volume grows, the time needed to go through all objects becomes unacceptable. Thus, we need to partition the data and access only the relevant partitions at query time. To do this in a generic metric space, we need to select some objects – we call them *pivots* – and using the distance between the pivots and the objects, we divide the collection. The two basic partitioning principles are called the *ball partitioning* and the *generalized hyperplane partitioning* [20] and they can divide a set of objects into two parts – see an example of ball partitioning in Figure 1a. Since the resulting partitions can be still too large, the partitioning can be applied recursively until all the partitions are small enough.

At query time, the triangular inequality property of the metric function is exploited to avoid accessing some partitions completely. All the remaining partitions are searched by the sequential scan. Even then, some distance-function evaluations can be avoided using the triangular inequality property again provided we have stored the distances between the pivots and the particular object. Since they are computed during the partitioning anyway and the distance is a real number, it incurs no additional computation and only a small extra space. We usually refer to this technique as the *pivot filtering* [8]. For a more detailed explanation see [20].

MESSIF Specification

One of the issues in the metric-space partitioning is the selection of pivots, since it strongly affects the performance of the query evaluation. There are several techniques [4] that suggests how to do the job and the framework provides a generic interface allowing to choose an arbitrary number of pivots from a particular collection (usually a bucket or a set of buckets). These pivots are usually selected so that effectiveness of a specific partitioning or filtering is maximized.

MESSIF Modules

- **Random pivot chooser** – this technique allows to randomly pick one or more objects from a collection of objects. It has $\mathcal{O}(1)$ complexity.
- **Incremental pivot chooser** – this algorithm [5] uses a so-called efficiency criterion that compares two sets of pivots and designates the better of the two. It has a quadratic complexity with respect to the size of the sample set.

- **On-fly pivot chooser** – remembers a predefined number of "best seen" pivots, which are updated whenever an object is inserted into a collection (usually a bucket). It has a linear complexity with respect to the objects inserted.

3.3 Metric Index Structures

The previous sections provide the background necessary for building an efficient metric index structure. We have the metric space objects with the distance function abstraction, we can process and store dynamic collections of objects using operations and we have tools for partitioning the space into smaller parts. Thus, to implement a working metric index structure we only need to put all these things together. Practically all algorithms proposed in the literature, see for example surveys [6, 13], can be easily built using MESSIF.

MESSIF Specification

The building of an index technique in MESSIF means to implement the necessary internal structures (the navigation tree, for instance) and create the operation evaluation algorithms. Since the buckets (where the data is stored) can evaluate operations themselves, the index must only pick the correct buckets according to the technique used and the actual internal state of the index. A MESSIF internal mechanism also automatically detect the operations implemented by an algorithm (the algorithms do not necessarily implement available operations) and also supports their parallel processing in threads.

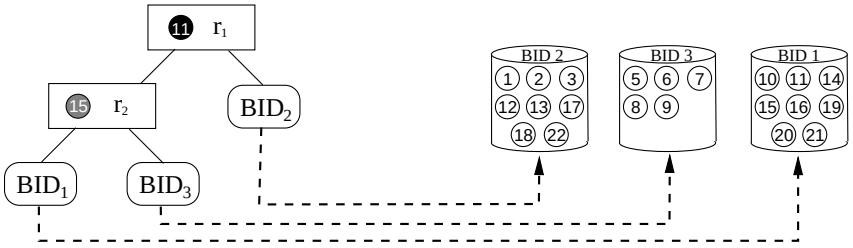

Figure 2: Example of Vantage Point Tree structure.

The simplicity of the implementation is demonstrated by the following example. A basic Vantage Point Tree (VPT) algorithm [20] builds a binary tree, where every internal node of the tree divides the indexed data into two partitions and objects are stored in leaves – see Figure 2 for an example, the space partitioning for this particular tree is shown in Figure 1a. In MESSIF, we need to implement the inner nodes, i.e. a data structure holding a pivot and a radius. Leaf nodes are the MESSIF buckets, so no additional implementation effort is needed. Then, the insert and range query operations are implemented. The navigation through inner tree nodes is quite easy, because it is only a condition-based traversal, and both the operations can be executed on leaf nodes using the standard MESSIF bucket implementation. The tree can be dynamic, thus splitting a leaf node can occur if its capacity limit is reached. Again, a MESSIF-driven split is easy: first, we need to choose a pivot for the new inner node using MESSIF pivot chooser. Then, a new bucket is allocated and the respective data is moved from the original bucket to the new one (this is a condition driven move, which is also an atomic function in MESSIF) – see Figure 1b.

MESSIF Modules

- **M-Tree** – a full implementation of the famous M-Tree indexing technique [7]. Implemented operations: insert, delete, range search, kNN search, incremental kNN search.
- **PM-Tree** – enhanced version of the M-Tree algorithm with additional pivot filtering [18]. Implemented operations: insert, delete, range search, kNN search, incremental kNN search.
- **D-Index** – hash-based metric indexing technique [9]. Implemented operations: insert, range search, kNN search.
- **aD-Index** – modified version of the D-Index technique featuring dynamically-adjusting navigation structures through linear hashing and multi-radii ball partitioning. Operations implemented: insert, delete, range search, kNN search.

- **VPT** – a basic implementation of simple dynamic metric index, which is provided as the implementation example above. Implemented operations: insert, range search.

3.4 Performance Measurement and Statistics

We have described the potential and the building blocks provided by the framework for creating index structures. However, essential part of every index is the performance statistics gathering. Statistics allow either automatic or manual tuning of the index and that can also serve during the operation-cost estimation (e.g. for a query optimizer). In the metric space, computation of the distances can be quite time demanding. Therefore, the time necessary to complete a query can vary significantly and it is also not comparable between different metric spaces. Thus, not only the time statistics should be gathered, but also the distance computations of various operations should be counted.

MESSIF Specification

Framework provides an automatic collecting of various statistics during the lifetime of an index structure – no additional implementation effort is needed. Any other statistics required by a particular index structure can be easily added. However, their querying interface is the same as for the automatic ones and they are accessible in the same way.

Specifically, every MESSIF module contains several global statistical measures. These are usually counters that are incremented whenever a certain condition occurs. For example, the *distance computations* counter is incremented when a metric function is evaluated. Moreover, other statistics can be based on the already defined ones – they can bind to an existing measure and then they will be updated every time the parent statistic is modified. For instance, a pivot-choosing technique usually needs to evaluate distances when selecting objects and we would like to know how many distance computations were necessary for that particular selection. Thus, the chooser binds its own *distance computations for pivot choosing* statistic to the existing *distance computations* counter from the metric module before the choosing begin. Then it computes the pivots normally and after the computation is finished it unbinds its statistics. In addition, the pivot chooser has its own statistics like the time spent while choosing pivots or the number of times the choosing was called.

Another very important issue is the statistics gathering during evaluation of operations, e.g. queries. Even though they can be executed simultaneously, MESSIF separates the respective statistics correctly – the concept of binding is used again, but the statistics are updated only locally within an operation.

MESSIF Modules

- **Query operation statistics** – the query response time, number of distance computations, buckets accessed and number of objects retrieved are automatically gathered for every query operation.
- **Bucket statistics** – numbers of inserts into a bucket, deletions and accesses are measured. Also the sum of statistics of all queries evaluated on the bucket is stored.
- **Algorithm statistics** – the algorithm records the sum of all the statistics from buckets and operations related to it and all additional index-specific statistics.

4 Distributed Data Structures

The huge amounts of digital data produced nowadays make heavy demands on scalability of data-oriented applications. The similarity search is inherently expensive and even though sophisticated dynamic disk-oriented index structures can reduce both computational and I/O costs, the similarity indexing approach requires some radical changes if a swift processing of large datasets is required.

The distributed computing provides not only practically unlimited storage capacity, but also significant augmentation of the computational power with a potential of exploiting parallelism during query processing. The Structured Peer-to-Peer Networks seem to be a suitable paradigm because of its inherent dynamism and ability to exploit arbitrary amounts of resources with respect to the size of the dataset managed by the system.

4.1 Computing Network

The objective of this part of the MESSIF platform is to provide the infrastructure necessary for the distributed data structures. The structured peer-to-peer networks consist of units (peers) equal in functionality. Each peer has a storage capacity and has access to the computational resources and to the communication interface. The concept of this basic infrastructure is depicted in Figure 3a.

The peer is identified by a unique address, and can communicate (using any underlying computer network) directly with any other peer whose address it knows. Peers can pose queries into the system. The request propagate arbitrarily through the system and some peers answer to the originator. The framework should provide support for this behaviour.

MESSIF Specification

The MESSIF networking operates over the standard internet or intranet using the family of IP protocols. Individual peers are identified by the IP address plus a port number. The entire communication is based on messages using the TCP and UDP protocols.

Generally, every message originated by a peer is sent to an arbitrary number of target peers. A receiver can either forward it further or reply to the originator (or both). If the sender expects receiving some replies it generally waits for a set of peers not known at the time message emission. To support this general message propagation algorithm, each message carries information about the peers it wandered through. As the reply messages arrive to the sender, it can use the message paths to manage the list of peers to await a reply from. The receiver is also able to recognize when the set of responses is complete.

As a message (carrying a query) propagates through the system, the visited peers can process some data before resending the message. The computational statistics (as defined in Section 3.4) of the processing at individual peers are collected automatically by the messages. After all replies arrive, the sender analyzes the statistics together with the message paths in order to calculate overall statistics of the entire message lifetime, for example: the *hop count*, the *total number of the message sending*, the *total number of distance computations* in all involved peers and the *parallel number of distance computations* – the maximal number of distance computations performed in a sequential manner during the message processing.

MESSIF Modules

- **Message** – the encapsulation of the data plus the routing information and collected statistics from the visited peers.
- **Message dispatcher** – encapsulates the peer's identification and usage of the TCP and UDP protocols; supplies the message sending service. It supports automatic UDP/TCP switching for maximizing the throughput, pooling of the TCP connections and the UDP multicast.
- **Reply receiver** – a service for waiting for the replies to a particular message emitted by this peer.

4.2 Distributed Metric Structures

The modules described in the previous section form the infrastructure necessary for building peer-to-peer data structures. These distributed systems are composed of *nodes* which consist of two logical components:

- data storage – data managed by a local index structure,
- navigation structure – a set of addresses of nodes in the system together with routing rules.

Figure 3b depicts such an overlay index and its utilization of the computing infrastructure. Every node is hosted by a physical computer that provides the CPU, the memory (volatile and persistent) and the network access. They maintain their local collections of objects that together form the overall data volume indexed in the distributed structure.

Any node can issue an operation (e.g. to insert an object) that is processed by the distributed index. The navigation mechanism is applied in order to route the operation to the nodes responsible for the respective data (e.g. a node holding the object's space partition). In these nodes, the operation is executed locally (e.g. the object is stored in the local data storage).

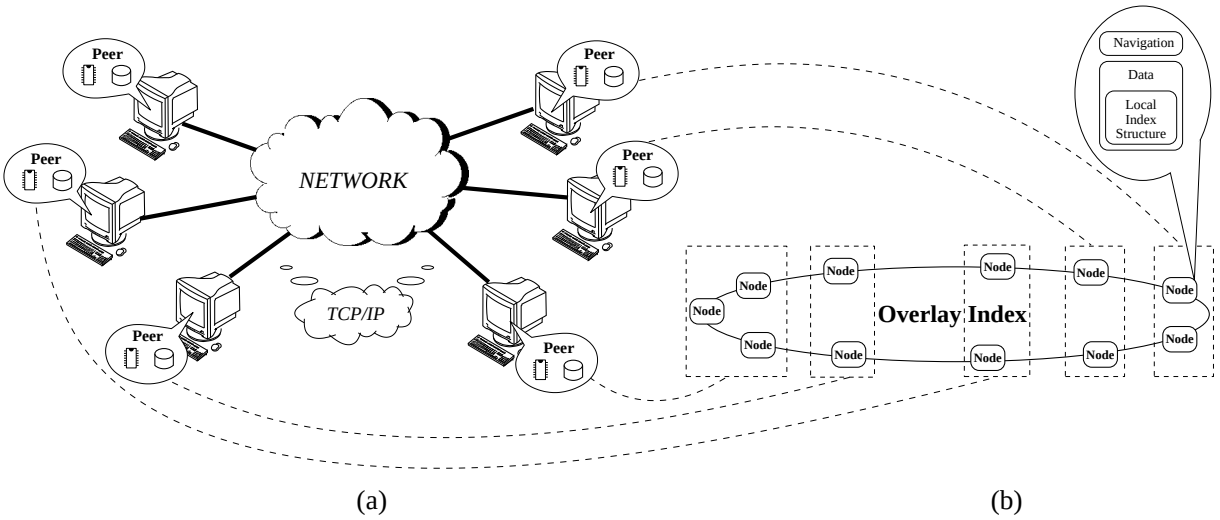

Figure 3: The computing network (a) provides infrastructure for overlay index (b).

MESSIF Specification

The realization of the node concept in MESSIF follows its logical structure. The node's data storage can employ buckets or any centralized indexes available as MESSIF modules as defined in Section 3.1 or 3.3, respectively. Therefore, the task of processing the operations locally is inherently available within the MESSIF functionality. The core of the distributed index is thus the navigation logic. It is responsible for determining the addresses of the nodes that should be reached by a specific operation. Once known, the MESSIF communication interface is used to contact the nodes and to wait for all the responses. The navigation needs not to be direct – several rerouting steps can occur at the contacted nodes. However, the essence of every distributed navigation algorithm is its ability to determine the correct address of the requested data. Generally, the distributed structures built using MESSIF are aimed developing metric-based similarity search, but also classical distributed hash table algorithms can be implemented.

MESSIF Modules

- ***GHT**** – the first metric-based distributed structure supporting the similarity search [2]. It uses the native generalized hyperplane partitioning of the metric space.
- ***VPT**** – a modification of the *GHT** structure for the metric ball partitioning principle [3].
- ***MCAN*** – an index structure that transforms the metric-space objects to a low-dimensional vector space, where the Content Addressable Network navigation algorithm can be used for similarity search [12].
- ***M-Chord*** – another transformation technique that simplifies the metric similarity search task to a *Chord*-powered distributed interval search in a simple number domain [16].
- ***Chord*** – a distributed hash table technique for a simple-key lookup [19].
- ***Skip Graphs*** – another distributed hash table technique for a key search [1].

5 Load Balancing for Similarity Searching

One of the issues considered in all structured peer-to-peer networks is *the balancing of the load*. In general, it is an automated process which reacts to variable load of the system and exploits available operations in order to keep the load distribution among the participating nodes as fair as possible. This effort can bring reduction of the query response time and increase the query-throughput of the system. Load balancing in the similarity-search structures has its specifics discussed in this section. The fundamental questions to ask are "How to define the load?" and "What are the operation available for balancing?".

Current load-balancing techniques define the load either as the data volume or as the number of

query-accesses per peer. However, evaluation of the metric distance function is typically computationally intensive and thus the processing of a similarity query is very time-consuming and may vary significantly for different queries at different nodes. Therefore, standard load definitions are not sufficient for our requirements and the node's load should be measured as the "computational load" of a given node.

Existing balancing strategies expect linearly-sorted and range-partitioned data which have the option of shifting a part of the data from a node to its *neighbor* and the option of splitting a node into two half-loaded nodes. Neither of these operations is available for the metric-based data structures. Therefore, MESSIF has a novel load-balancing framework that suits the requirements and potential of peer-to-peer structures for similarity search in metric spaces.

MESSIF Specification

The most common source of a node's overloading is large data volume stored at the node. Part of the data must be moved to another node. In a general metric-based structure, the only data-movement operation is a split of the node. Therefore, MESSIF separates the logical layer of the distributed structure from the physical infrastructure – several logical nodes can be hosted by one peer. The nodes then share the computational and storage resources and the communication interface of the peer.

Furthermore, the MESSIF load balancing utilizes the concept of node replication. A replicated node can forward a search query to a selected replica (hosted by another peer) instead of processing the query locally.

The load-balancing mechanism treats the set of peers as fixed and exploits the following balancing operations that affect the logical layer of the particular system:

split splits the data of a node (equally, if possible) and creates a new node at another peer;

leave removes a node from the system – either merges the node with some other node (if possible) or re-inserts the data to the system after a node deletion;

migrate moves a node to another peer – typically a less loaded one;

replicate creates a replica of the specified node at another peer;

unify removes a specific replica of a given node.

The MESSIF balancing system analyzes the computational load of a node precisely in order to select the best balancing action when an overload occurs. In general, processing of a query at a node may be too time-consuming either since evaluation on the local data takes too much time or a query is waiting until the processing of other operations finishes. According to this observation, the load is measured as two separate quantities: **proc_load** represents the average costs of the local processing of a query operation and **wait_load** represents the average time a query is *waiting* before processed.

Each peer monitors its load values and compares them with the current average load values in the whole system. It reacts to overloading according to specific balancing rules which follow these basic principles: If the local processing of a query at a given node is too long then the node must split. If the frequency of queries hitting this peer is too high and thus the **wait_load** is too large then some node can be migrated to other peer or – if there is only one node at this peer – the node is replicated.

The MESSIF load-balancing system uses a specific distributed peer-to-peer algorithm [14] based on gossips to maintain approximations of current system-average load values. The algorithm exploits standard messages (query, insert, management) to exchange the data and increases the load of the network only negligibly. The messages also carry information about low-loaded peers in the system. Figure 4 summarize the work of a balancing module at every peer. For more details on the sketched load-balancing system see a separate paper [15].

MESSIF Modules

- **Peer** – frame for logical nodes of a distributed metric structure; manages the load statistics and provides the communication interface to all nodes.
- **Gossip module** – management of the gossiping algorithm and current global-average load values.
- **Balancing module** – encapsulates the load balancing strategies and execution of balancing actions when an overload occurs.

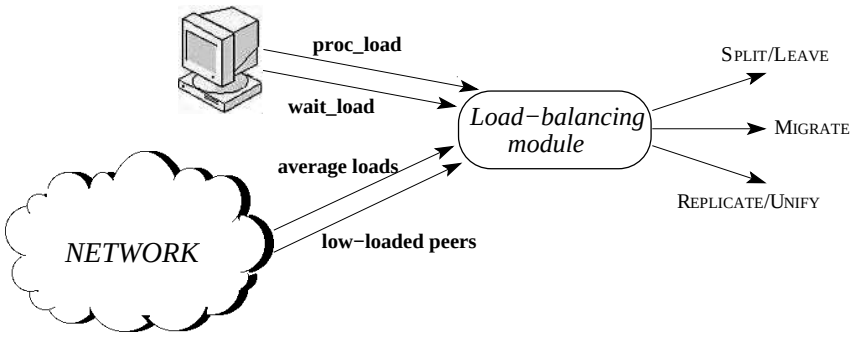

Figure 4: Work schema of the load-balancing module.

6 Multi-Feature Similarity Search

In the previous, we have outlined the MESSIF-based index structures design. Thus, we are able to answer basic similarity queries using a well-defined interface, regardless of what algorithm is used behind. This can be advantageous for our next goal – the multi-feature similarity searching, where several similarity sub-queries are executed on different features of the objects stored in the database. The obtained sub-results are then combined to compute the overall similarity of the respective object. Let us imagine we have a database of images from which we can extract their shape and color features forming two different metric spaces with shape-distance and color-distance functions. Then we would like to answer complex similarity queries like "give me all red-most circle-like objects", because we can retrieve the red-similar objects in color-feature metric space and the circle-similar objects in the shape features. Of course, we need to combine the results from both the similarity queries – the red-most object can be the less circle-like or vice versa. This is done using the aggregate similarity function and we can sort the retrieved objects according to its results.

More formally, let $\mathcal{D}^m$ be a domain of multi-feature objects $o = (o^1, o^2, ..., o^m), o^1 \in \mathcal{D}^1, o^2 \in \mathcal{D}^2, \ldots, o^m \in \mathcal{D}^m$. There is a metric function d^i defined for every feature and the pairs $(\mathcal{D}^i, d^i)$, $i = 1, \ldots, m$ form respective metric spaces. To compute the overall similarity, the user specifies an m-ary aggregate function t, which must be monotonous. For a given query object q with features $q = (q^1, q^2, ..., q^m)$, the overall similarity between q and any object o can be computed as $t(d^1(q^1, o^1), d^2(q^2, o^2), ..., d^m(q^m, o^m))$. The result of function t should be a non-negative real number and it expresses the similarity in the same way as metric functions – the higher the value the more dissimilar the objects are.

To retrieve the k best-matching objects in multiple features, there were proposed two generic algorithms – Fagin's $\mathcal{A}_0$ Algorithm [10] and the threshold algorithm [11]. Since the threshold algorithm enhances the $\mathcal{A}_0$ and since it stops earlier in most situations, we will focus on that one only. It works as follows.

An incremental nearest neighbor search is initialized in every feature metric space for the respective query feature q^i. Then, every iteration consists of the following steps.

1. Next object (using the incremental nearest neighbor) is retrieved for every feature $i = 1, \ldots, m$.
2. Let d^i_{max} be the distance of that next object in feature i, then the *threshold value* $\theta = t(d^1_{max}, ..., d^m_{max})$.
3. For every object from the first step, all distances for all other features are evaluated and the overall similarity is computed using t.

The iterating stops if at least k objects with overall similarity lower then or equal to θ have been obtained.

MESSIF Specification

The general threshold algorithm can be implemented using the MESSIF building blocks. The incremental nearest neighbor operation is supported by the framework. The framework can also serve to hold different features of an object in different metric spaces – we can link the different features of the same objects using the same OID. However, some additional support is needed for hosting multiple index structures for different features and for the generalized aggregate similarity function. Currently, this is a work in progress, and the design is as follows.

A separate index overlay is built for every feature of the stored objects. Object identifiers OID are unique for any distinct object stored, but they are shared among the overlays. Whenever a new object is stored into the whole structure, it is assigned the OID automatically. Then, the respective features of the object are separately inserted into respective overlays. Therefore, all the overlays hold exactly the same number of object features with exactly the same OIDs.

Moreover, a special primary-key search overlay is established to store all the features of every object. Its search capabilities are different – the objects are indexed using their OIDs. Therefore, we can retrieve all the features for the provided set of OIDs, which is vital for the third step of the threshold algorithm. Of course, when a new object arrives, it is also inserted into this overlay according to its OID.

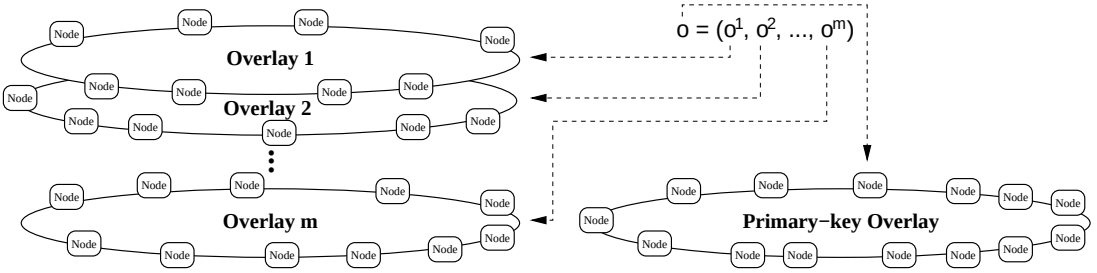

Figure 5: Similarity overlays and primary-key overlay.

The whole architecture is depicted by Figure 5. Specific object features are indexed in the respective overlays and all the features are stored together in the primary-key overlay. The search algorithm then follows the aforementioned threshold algorithm with some slight modifications. First, the incremental nearest neighbor search retrieves more than just one object in one iteration. Also, all nearest neighbor searches from the first step run in parallel. Once completed, the threshold value is updated, since we know the maximal distances in respective features. Then, using the primary-key overlay, all the retrieved objects' features distances are computed in parallel. Only the threshold values of the objects are returned back, they are sorted and the threshold is checked for stop condition. If the algorithm finishes, we have the sorted list of best matching objects. Otherwise, additional nearest neighbors are executed, retrieving more objects and the procedure is repeated.

There are still some research challenges – for example, the number of objects retrieved by the incremental nearest neighbor query can be estimated using some heuristics, we can also save some queries on the primary-key overlay (using the known maximal distances), etc. Also, every aforementioned overlay can be load-balanced and we can tailor the specific index structures to every overlay.

MESSIF Modules

- **Multi-Overlay Dispatcher** – a peer coordinator for hosting several nodes of different index overlays on the same peer. Can be used to access a specific feature index structure and to execute an operation on it.
- **Multi-Feature Query Operation** – operation encapsulating the threshold algorithm. It uses the **Multi-Overlay Dispatcher** to execute object's feature incremental nearest neighbor queries.

7 MESSIF User Interfaces

Finally, let us briefly mention the interfaces that can be used by the user to control the index structures built using MESSIF. Since all the structures use the same interface, we can control them uniformly. The designer can annotate the published parts of the newly implemented structure to provide a unified description and help texts. For example, one of the initialization parameters of the M-Tree structure is the size of its inner nodes. Since it is correctly annotated, the user will be provided with the option of specifying the parameter. The MESSIF can also automatically detect all the operations that are supported by a specific algorithm offering the user to work with them. Very important is also the ability of the MESSIF to gather all sorts of statistics, which are available for the users too.

The MESSIF provides several user interfaces, capabilities of which vary from the most simple ones allowing the user to write commands to a simple prompt to the complicated graphical client, which offers

comfortable manipulation with the running structure along with an easy-to-read statistics presented by graphs. In a distributed environment, the user interface (client) can connect to running peers to control them. The MESSIF also includes a special monitoring interface for distributed environment, allowing to show status of the running peers.

MESSIF Modules

- **Batch-Run Interface** – a text-file driven batch that can initialize a structure, run operations on it (inserts, queries, etc.) and log the results including statistics into text files.
- **Telnet User Interface** – a simple command prompt interface available through a TCP connection.
- **Web User Interface** – a graphical applet able to connect to a running index structure.
- **Window User Interface** – a Java GUI application with wizards for initializing a structure and executing operations. It also contains a plotting unit for displaying statistics as graphs.
- **Web Peer Monitor** – web application available for monitoring the status of a distributed peer-to-peer network.

8 Conclusions

The similarity search based on the metric space data-model has recently become a standalone research stream, which arouses greater and greater interest. The number of data structures and indexing techniques for metric data increases and an important part of such a research is a prototype implementation and subsequent experimental evaluation of the structure's performance.

Since we concern ourselves with this research area, we felt the need for a development platform that would make the implementation easier. It should also support sharing and reusing of the code and make the testing more efficient and comparison of the results more fair. In this paper, we have presented MESSIF – The Metric Similarity Search Implementation Framework, which provides the following:

- encapsulation of the metric space concept – developers can use the data objects transparently regardless of the specific dataset – new data types can be added easily;
- concept of operations – introducing a uniform interface to modify and query the data;
- management of the metric data – storing objects in buckets with automatic evaluation of basic similarity queries and buckets-splitting based on the metric indexing principles;
- automatic performance measurement and collecting of various statistics including a uniform interface for accessing and presenting the results;
- communication layer for distributed data structures – message navigation, automatic collecting and merging of statistics;
- specialized load-balancing system for distributed index overlays;
- support for complex similarity queries in multi-metric spaces;
- user interfaces – designed to control both the centralized and the distributed data structures.

A number of data structures have been implemented using the MESSIF. Their implementations share common code basis and can effectively utilize any other as subsystems, e.g. individual nodes of a distributed system can exploit any centralized structure for indexing and searching its local data. The use and the testing of all systems based on the MESSIF is boosted up by having a unified system control, input format of data, and output format of query results and statistics values.

The MESSIF is becoming the basic implementation infrastructure for a new European project SAPIR. We would gladly provide the MESSIF source codes for research purposes upon request.

References

[1] James Aspnes and Gauri Shah. Skip graphs. In *Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 384–393, January 2003.

[2] Michal Batko, Claudio Gennaro, and Pavel Zezula. Similarity grid for searching in metric spaces. *DELOS Workshop: Digital Library Architectures, LNCS*, 3664/2005:25–44, 2005.

[3] Michal Batko, David Novak, Fabrizio Falchi, and Pavel Zezula. On scalability of the similarity search in the world of peers. In *Proceedings of INFOSCALE 2006, Hong Kong, May 30 – June 1, 2006*, pages 1–12, New York, NY, USA, 2006. ACM Press.

[4] Benjamin Bustos, Gonzalo Navarro, and Edgar Chávez. Pivot selection techniques for proximity searching in metric spaces. In *SCCC 2001, Proceedings of the XXI Conference of the Chilean Computer Science Society*, pages 33–40. IEEE CS Press, 2001.

[5] Benjamin Bustos, Gonzalo Navarro, and Edgar Chávez. Pivot selection techniques for proximity searching in metric spaces. *Pattern Recognition Letters*, 24(14):2357–2366, 2003.

[6] Edgar Chávez, Gonzalo Navarro, Ricardo A. Baeza-Yates, and José Luis Marroquín. Searching in metric spaces. *ACM Computing Surveys (CSUR)*, 33(3):273–321, September 2001.

[7] Paolo Ciaccia, Marco Patella, and Pavel Zezula. M-tree: An efficient access method for similarity search in met ric spaces. In *VLDB'97, Proceedings of 23rd International Conference on Very Large Data Bases, August 25-29, 1997, Athens, Greece*, pages 426–435. Morgan Kaufmann, 1997.

[8] Vlastislav Dohnal. *Indexing Structures fro Searching in Metric Spaces.* PhD thesis, Faculty of Informatics, Masaryk University in Brno, Czech Republi c, May 2004.

[9] Vlastislav Dohnal, Claudio Gennaro, Pasquale Savino, and Pavel Zezula. D-Index: Distance searching index for metric data sets. *Multimedia Tools and Applications*, 21(1):9–33, 2003.

[10] Ronald Fagin. Combining fuzzy information from multiple systems. In *Proceedings of the Fifteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, June 3-5, 1996, Montreal, Cana da*, pages 216–226. ACM Press, 1996.

[11] Ronald Fagin, Amnon Lotem, and Moni Naor. Optimal aggregation algorithms for middleware. In *PODS '01: Proceedings of the twentieth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 102–113, New York, NY, USA, 2001. ACM Press.

[12] Fabrizio Falchi, Claudio Gennaro, and Pavel Zezula. A content-addressable network for similarity search in metric spaces. In *Proceedings of the 3rd International Workshop on Databases, Information Systems, and Peer-to-Peer Computing, Trondheim, Norway, 2005*, pages 126–137, 2005.

[13] Gísli R. Hjaltason and Hanan Samet. Index-driven similarity search in metric spaces. *ACM Transactions on Database Systems (TODS'03)*, 28(4):517–580, 2003.

[14] David Kempe, Alin Dobra, and Johannes Gehrke. Gossip-based computation of aggregate information. In *FOCS '03: Proceedings of the 44th Annual IEEE Symposium on Foundations of Computer Science*, page 482, Washington, DC, USA, 2003. IEEE Computer Society.

[15] David Novák. Load balancing in peer-to-peer data networks. In *MEMICS 2006, 2nd Doctoral Workshop on Mathematical and Engineering Methods in Computer Science*, pages 151–157, Brno, Czech Republic, 2006. Faculty of Information Technology, Brno University of Technology.

[16] David Novak and Pavel Zezula. M-Chord: A scalable distributed similarity search structure. In *Proceedings of First International Conference on Scalable Information Systems (INFOSCALE 2006), Hong Kong, May 30 – June 1, 2006*, pages 1–10, New York, NY, USA, 2006. ACM Press.

[17] Yossi Rubner, Carlo Tomasi, and Leonidas J. Guibas. A metric for distributions with applications to image databases. In *ICCV '98: Proceedings of the Sixth International Conference on Computer Vision*, page 59, Washington, DC, USA, 1998. IEEE Computer Society.

[18] Tomáš Skopal. Pivoting M-tree: A metric access method for efficient similarity search. In *Proceedings of the Dateso 2004 Annual International Workshop on DAtabases, TExts, Specifications and Objects, Desna, Czech Republic, April 14–16, 2004.* Technical University of Aachen, 2004.

[19] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of ACM SIGCOMM*, pages 149–160, San Diego, California, United States, 2001. ACM Press.

[20] Pavel Zezula, Giuseppe Amato, Vlastislav Dohnal, and Michal Batko. *Similarity Search: The Metric Space Approach*, volume 32 of *Advances in Database Systems.* Springer-Verlag, 2006.