ADAM — A Database and Information Retrieval System for Big Multimedia Collections

Ivan Giangreco Ihab Al Kabary Heiko Schuldt
Department of Mathematics and Computer Science
University of Basel, Switzerland
{*firstname.lastname*}*@unibas.ch*

***Abstract*—The past decade has seen the rapid proliferation of low-priced devices for recording image, audio and video data in nearly unlimited quantity. Multimedia is Big Data, not only in terms of their volume, but also with respect to their heterogeneous nature. This also includes the variety of the queries to be executed. Current approaches for searching in big multimedia collections mainly rely on keywords. However, manually annotating every single object in a large collection is not feasible. Therefore, content-based multimedia retrieval –using sample objects as query input– is increasingly becoming an important requirement for dealing with the data deluge. In image databases, for instance, effective methods exploit the use of exemplary images or hand-drawn sketches as query input. In this paper, we introduce ADAM, a novel multimedia retrieval system that is tailored to large collections and that is able to support both Boolean retrieval for structured data and similarity-based retrieval for feature vectors extracted from the multimedia objects. For efficient query processing in such big multimedia data, ADAM allows the distribution of the indexed collection to multiple shards and performs queries in a MapReduce style. Furthermore, it supports a signature-based indexing strategy for similarity search that heavily reduces the query time. The efficiency of ADAM has been successfully evaluated in a content-based image retrieval application on the basis of 14 million images from the ImageNet collection.**

I. Introduction

With the proliferation of ubiquitous devices for digitally capturing and recording image, audio and video data, multimedia retrieval has entered the Big Data arena. The tremendous growth of multimedia data has put serious challenges on the systems handling these data. This includes the efficiency requirements multimedia retrieval systems are expected to provide to the user. Moreover, the sheer size of multimedia collections and the heterogeneous nature of both the data and the queries make storing, organizing, and retrieving multimedia data a difficult task to undertake.

A researcher could approach the problem of building a multimedia retrieval system from two different sides. One option is the use of a traditional relational database for storing the data. Traditional databases (DB) provide excellent support for structured data and queries that adhere to *Boolean predicates*. However, they usually do not provide a viable solution for query paradigms such as similarity retrieval, not only because DB systems have no notion of querying unstructured data (such as full texts or multimedia objects), but particularly because they support similarity search and partial matches only in a very limited way. As a second option, the researcher might think of using an information retrieval (IR) system. IR systems provide similarity search, for instance by relying on sample objects to find similar objects. The results to a *similarity query* oblige a strict ordering based on a score that represents the element's relevance with respect to the query. However, an IR system lacks valuable database features for exact search in meta data, such as index structures, query optimization, etc. Furthermore, multimedia IR systems have not yet entered the field of Big Data. To date, search systems for multimedia data are mainly built upon monolithic storage systems that are tailored to the application using the data.

Ultimately, both types of systems have their drawbacks, especially when considered in settings where a user might want the combination of various query paradigms at high efficiency and at a large scale. While a DB system is catered to structured data and Boolean predicates in large scales, it is not useful for similarity queries in unstructured data. IR systems, in contrast, support similarity queries, but often lack sophisticated index structures for searching in meta data and, thus, do not scale well.

The contribution of this paper is twofold: First, we introduce ADAM, a novel approach to seamlessly combine database technology and information retrieval for big multimedia data. ADAM brings the ability to handle and store any type of multimedia data and their corresponding features. As soon as a developer has specified an algorithm for extracting intrinsic features of the multimedia objects, the ADAM system is able to take over the execution of the extraction task, the storage and the indexing of the feature data and the multimedia objects. To this end, ADAM is based both on the relational database model and the vector space model which considers multimedia objects as vectors in a high-dimensional feature space and defines the similarity between two objects by the distance in the spanned space. For structured data, ADAM can query using Boolean filtering predicates by making use of traditional B-tree index structures; for ranking the elements of a collection according to a similarity score, a similarity retrieval can be performed. The ADAM system

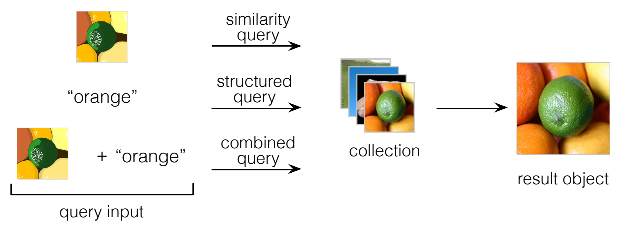


Figure 1. Sample retrieval use case for an image retrieval application.

not only combines both query paradigms but also focuses on big multimedia collections by distributing data over multiple shards so that they can be queried in a MapReduce style and by efficiently handling similarity queries in large volumes of data using a signature-based index structure on the single shards. Second, the paper introduces a fully working, general purpose implementation for multimedia data that builds on these concepts. We demonstrate the effectiveness and efficiency of ADAM in the context of a content-based image retrieval application that exploits hand-drawn sketches as query inputs. The application allows users to search in image collections using keywords, e.g., for searching in the tags of an image, and/or hand-drawn sketches, i.e., for searching in the multimedia objects itself. We summarize the query paradigms supported by ADAM in the setting of the exemplary image retrieval application in Figure 1. The application was used in the evaluation based on 14 million images from the ImageNet collection. The evaluations have shown ADAM's excellent scalability characteristics. Due to the inherent implementation of feature extraction and search in a MapReduce style, even collection sizes beyond 14 million objects can be dealt with efficiently.

This paper is structured as follows. Section II discusses related work. In Section III, we present the components and the concepts that underly ADAM. Details of the implementation are given in Section IV. In Section V, we evaluate ADAM in the context of a content-based image retrieval application. Section VI concludes this paper.

II. Related Work

Early work in Garlic [1] integrates multiple federated databases into one distributed system for multimedia data. The system is based on an object-oriented database model that is exploited to store the multimedia objects. For query formulation, the authors extend the object-oriented query language (OQL). Similarly, [2] introduces the system Chabot, an IR system using a PostgreSQL database for storing extracted features. The authors implement complex types, user-defined indices and user-defined functions into the database to support IR data types and queries. Chabot not only supports text-based queries, but also allows to improve results by providing content-based queries (e.g., "images with some orange in it"). Both Garlic and Chabot only support Boolean predicates rather than similarity-based queries. In [3], the authors introduce Mirror, a database supporting content-based multimedia retrieval. The authors describe the engineering factors for creating a multimedia IR-DBMS and introduce Moa, a new relational algebraic framework based on the non-first normal form (NF^2). Mirror is implemented on top of the object-relational DBMS Monet. DISIMA DBMS [4] is a DBMS that allows to store syntactic features (e.g., color, shape) and semantic features (i.e., real world objects) in an object-oriented data model. The system supports content-based searches and searches on image semantics. The authors implement an extended version of OQL for multimedia objects (MOQL) and VisualMOQL, a visual counterpart to MOQL. To increase the performance of the system, the authors use three-dimensional extendible hashing that allows to pre-filter images based e.g., on the average color. The authors in [5] introduce a system that combines low-level (syntactic) features with semantic features in an object-relational Oracle 11g database. The database is extended by several User Defined Types (UDT) following the MPEG-7 standard descriptors, and operations implemented in PL/SQL, e.g., to evaluate similarity measures. Recent work in [6] makes use of the MapReduce paradigm for querying large sets of image data in a cloud environment. The authors use extended Cluster Pruning for indexing and port it to the MapReduce paradigm on the Hadoop platform.

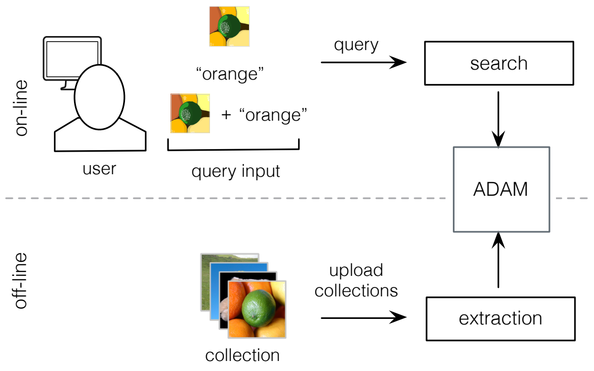


Figure 2. Two phases of an exemplary image retrieval system.

In the 1980s and 90s, the field of databases has seen a large variety of indexing methods for vector data. Many tree-based indexing structures, including the family of R-trees [7], the M-tree [8], the X-tree [9], etc. were introduced. However, all tree-based approaches suffer from the curse of dimensionality: [10] shows that with increasing dimensionality every tree-based search structure degenerates to a sequential search. The authors claim that for indexing purposes tree structures are generally outperformed by a simple sequential scan if the number of dimension exceeds around 10. [10] introduces the Vector Approximation (VA) File approach, a method based on signatures for improving efficiency in processing feature vectors. Similar approaches can be found, e.g., in iGrid [11] and BitMatrix [12].

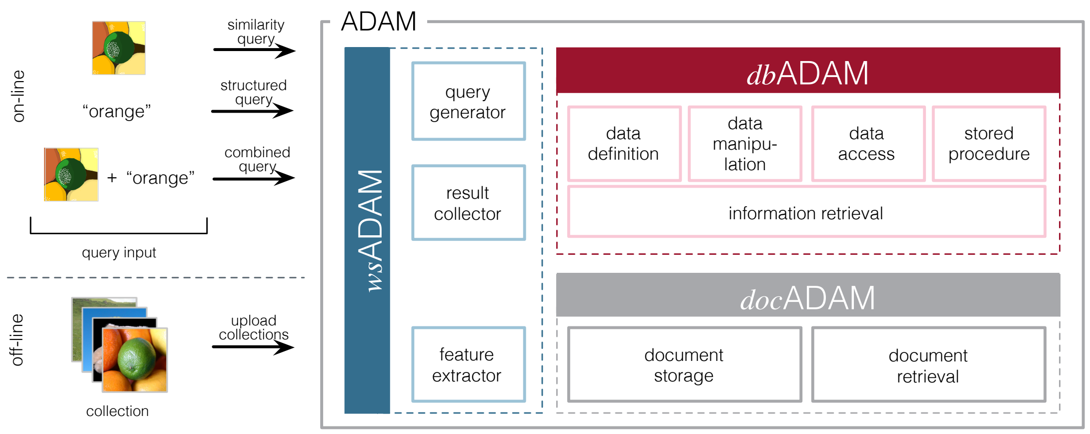


Figure 3. System view of ADAM.

III. ADAM: System Model

In the following, we detail the architecture and the concepts that underly ADAM. Information retrieval systems are often divided into an off-line part, i.e., the feature extraction, and an on-line part, i.e., the query phase (see Figure 2). In both phases, we hand over the responsibility for the storage, organization and retrieval of multimedia objects to ADAM.

Figure 3 gives a broad overview of the system architecture. The middleware *ws*ADAM acts as a gate to all functions of ADAM. It is responsible for the distribution of new data (at feature extraction-time) and –in interplay with *db*ADAM– the orchestration of queries (at query-time). *db*ADAM is the core of the system that stores and retrieves documents from the database. The ADAM system can be scaled to larger collection sizes by adding new *db*ADAM shards that act as workers for the *ws*ADAM layer. Finally, *doc*ADAM is responsible for storing and retrieving the actual multimedia documents (or excerpts of it) for display purposes.

A. Components

1) wsADAM: *ws*ADAM constitutes a middleware layer acting as an entry point for access to any other component of ADAM. It assumes in both phases of an IR system (on- and off-line) the role of central orchestrating component. Despite the orchestrating nature of *ws*ADAM, this component can be distributed to multiple sites which can all act separately as a gate to the ADAM system.

At feature extraction-time, i.e., the *off-line* part of an information retrieval system, *ws*ADAM is responsible for the feature extraction and distribution of new multimedia objects to one or multiple *db*ADAM shards. The extracted features are then distributed –ensuring an even distribution– to one or multiple *db*ADAM servers and stored there. The multimedia object itself is stored on the *doc*ADAM servers.

At query-time, i.e., the *on-line* part of an IR system, the middleware layer *ws*ADAM intercepts incoming queries and distributes these to all *db*ADAM shards in a MapReduce fashion. *ws*ADAM acts as a broker that distributes (map) and orchestrates the sub-queries running on the separate nodes, collects their results and combines these to one single, result set (reduce) that is returned to the query initiator.

2) dbADAM: *db*ADAM is responsible for storing and retrieving the extracted feature vectors and additional meta data. At its core, *db*ADAM is an adapted relational database using an extended SQL dialect: It supports a new data type for storing the features extracted from the multimedia object. For querying the features, *db*ADAM provides language constructs to support k nearest-neighbour retrieval. In addition, similarity queries can be enriched to take Boolean predicates to further limit the results. *db*ADAM allows to use union or intersect operations for single-/multi-feature single-/multi-object queries (as defined by [13]) by applying distance combining functions that are based on fuzzy logic. To increase the performance of similarity retrieval in big multimedia collections, *db*ADAM implements Vector Approximation indexing. Finally, *db*ADAM supports specialized function-creation methods for defining distance functions that can be used in similarity queries to calculate the similarity between two multimedia objects.

*db*ADAM knows the following additional SQL statements:

- `FEATURE` data type to store extracted features
- adjusted `SELECT` statement that allows to perform a similarity-based retrieval
- adjusted `UNION` and `INTERSECT` operations with options `CRISP` or `STANDARD` for distance combination
- `CREATE VA` and `DROP VA` with option for `EQUIDISTANT` and `EQUIFREQUENT MARKS`
- `CREATE`, `ALTER` and `DROP` commands for `DISTANCE` functions to create custom distance functions

The number of *db*ADAM shards to deploy to the ADAM system depends largely on the collection size and performance requirements. By adding additional *db*ADAM shards the local collection size on each shard can be reduced with overall improvements in efficiency.

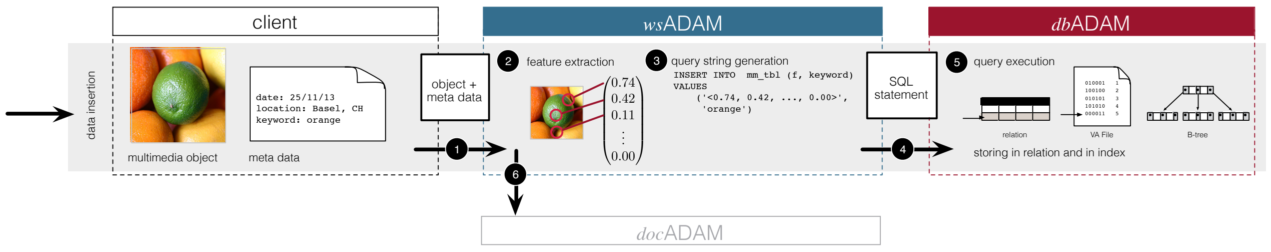


Figure 4. Data insertion process in ADAM.

B. Concepts

1) Data Definition and Data Insertion: The creation of a collection within ADAM necessitates that the user specifies various elements: First, one or multiple feature extraction algorithms have to be added to *ws*ADAM. These could include extraction algorithms for local features, such as SIFT [14], and extraction algorithms for global features, such as the dominant color descriptor. The feature extraction algorithms are applied in the off-line phase for extracting features from new multimedia objects added to the collection and at on-line time for performing the feature extraction on query objects (e.g., to extract the features from a query sketch) used for retrieval. Second, the relation that is used to store the feature data and the corresponding meta data has to be defined. Optionally, custom distance functions can be created for use at retrieval time. Finally, indices can be added to the relation to increase the efficiency of query processing in the on-line phase of the system.

To insert data into a relation, a new multimedia object is given together with predefined, structured meta data to *ws*ADAM (1). *ws*ADAM uses the available feature extraction algorithms to extract the features of the new object (2). It generates a query string (3) that is sent to one of the *db*ADAM shards (4) to store the extracted feature vector and the meta data (5). The multimedia object itself is stored on *doc*ADAM (6). Figure 4 summarizes the insertion process.

2) Retrieval: ADAM supports both Boolean retrieval and approximate similarity search. Boolean retrieval can be applied on all structured fields, i.e., the meta data corresponding to an object (such as the content creator, the creation time, etc.). Similarity-based retrieval, on the other hand, can be used for searching within the extracted feature vectors for k similar documents (in a k nearest neighbor search).

Text retrieval systems have known the problem of *source selection* very well, i.e., how to select only a subset of shards that is able to answer a given query [15]. Multimedia retrieval has currently no solution to this problem. Thus, queries must be posed to all shards in the system, if no other (Boolean-based) selection mechanism is present. In ADAM, queries are sent to all available shards and processed in a MapReduce fashion. Figures 5 and 6 illustrate the processing of a query in a deployment and a behavior view, respectively.

When a query reaches *ws*ADAM (1), the query extractor extracts (2) the necessary features from the incoming query object (e.g., a sketch) and that is used by the query generator to generate a query string (3). *ws*ADAM sends the generated query string to all *db*ADAM shards (4). Considering the MapReduce paradigm, this step corresponds to the *map* part: the query received by the query master in *ws*ADAM is sent to all *db*ADAM shards that constitute the workers. The *db*ADAM shards use the query string to perform a local lookup on their content (5) and return their local results (6) to the orchestrating query master. The result collector of *ws*ADAM retrieves the results and constructs a single, coherent result set (7) in a *reduce* step. For the reduction, the lists containing the subresults from each shard are merged and sorted according to the distance measure. The results are then returned to the query initiator (8). Subsequently, the client contacts *ws*ADAM to retrieve from *doc*ADAM the documents to display (9).

3) Indexing: For improving query efficiency, ADAM supports an adapted version of the Vector Approximation (VA) File as introduced in [10]. The idea of VA indexing is to compress using a quantization approach feature vectors to a simple short signature and later query the signatures in a sequential manner. To create a VA signature, a fixed-length bit string for each data point is generated. For that purpose, the data space is divided in $2^{b_{tot}}$ cells, where b_{tot} denotes the total length of the bit-signature, and the cells are enumerated in a binary way. Each dimension receives b_d bits that are finally concatenated to create the full bit mask [10].

To constitute the cells, ADAM provides the user two different strategies, i.e., an `EQUIDISTANT` strategy, in which all cells have the same size in the feature space, and an `EQUIFREQUENT` strategy that builds up the cells by sampling from the data space and creating cells that contain approximately the same amount of data points.

At query time, in a first *prefiltering* phase, the VA index is used to perform a coarse filtering based on the bit string. Using the VA signature, an upper and lower bound of the distance can be calculated with very few simple calculations. The bounds are used to determine whether the data object is within boundaries that would make it come into consideration for the final results. Because of the compactness of the VA signatures, only a small amount of

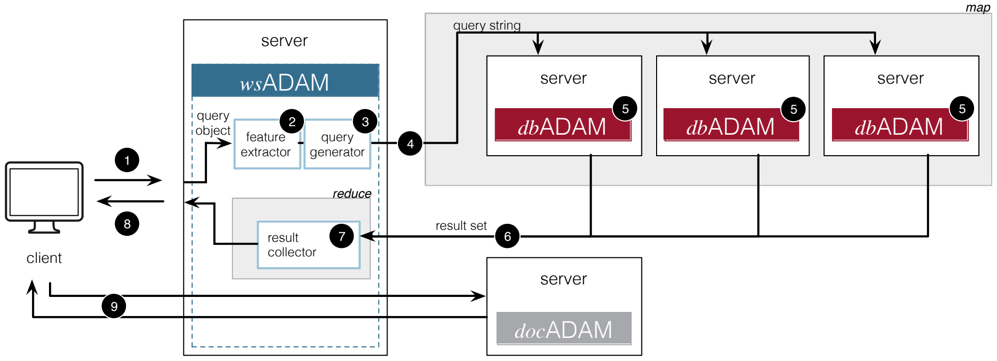
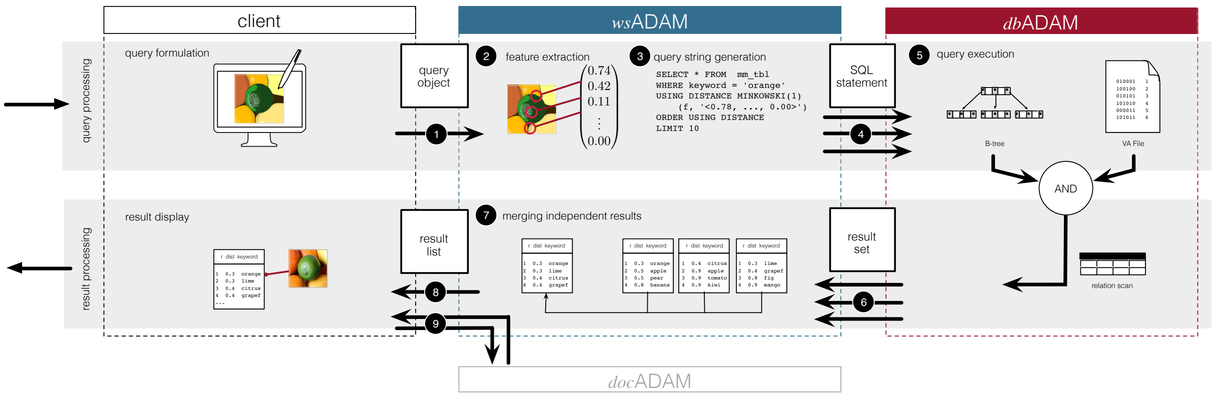


Figure 5. Query processing in ADAM with focus on the deployment.



```
SELECT * FROM  mm_tbl
WHERE keyword = 'orange'
USING DISTANCE MINKOWSKI(1)
    (f, '<0.78, ..., 0.00>')
ORDER USING DISTANCE
LIMIT 10
```

Figure 6. Query processing in ADAM with focus on the system behavior.

pages have to be loaded to read the full index. In the second phase, the *refinement* phase, if the multimedia object has not been filtered out in the first phase already, the exact distance is calculated by loading the full feature vector. Thus, the full and exact calculation of the distance has not to be processed for all data points, avoiding unnecessary data page accesses and full distance computations while at the same time not deteriorating the quality of the results [10].

IV. Implementation

A. *ws*ADAM

We have implemented *ws*ADAM as a Java web service that can be deployed to multiple nodes. On the one hand, it is responsible for the off-line feature extraction. On the other hand, at query time, it receives JSON messages from the clients and builds according to its own rules the query string to send to the *db*ADAM instances.

*ws*ADAM uses the MapReduce paradigm to process a query. Our implementation follows very much the ideas as suggested in [16]: *ws*ADAM acts as a master that orchestrates the query executed on all *db*ADAM shards. Then, *ws*ADAM is responsible for creating a coherent result set that is returned to the query initiator.

B. *db*ADAM

*db*ADAM is implemented in PostgreSQL 9.3 and is, thus, based on the object-relational data model. We have extended PostgreSQL to support the storage of feature vector data using a new `FEATURE` data type. Further, we have extended the `SELECT` statement to support similarity-based queries by applying a distance function on the feature vectors. A `SELECT` statement in *db*ADAM generally looks as follows:

```
SELECT * FROM mm_tbl
WHERE keyword = 'orange'
USING DISTANCE MINKOWSKI(1)
    (f, '<0.78,...>')
ORDER USING DISTANCE
LIMIT 10;
```

in which the traditional `WHERE` statement is used for performing a Boolean retrieval and the `USING DISTANCE` statement is used to denote a similarity retrieval. We have adapted the query plan generator to efficiently combine Boolean and similarity retrieval. Furthermore, we have added several useful functions to *db*ADAM such as the calculation of the Minkowski distances, as shown in the example. The query vector is denoted in the example using `'<...>'`, which is the way ADAM specifies feature vectors.

V. Performance Results

A. Experimental Setup

We have evaluated ADAM in the context of a content-based image retrieval (CBIR) application that exploits hand-drawn sketches to search in a collection of images for similar objects. The application uses angular radial partitioning (ARP) [17] to constitute 4×4 partitions in an image. Using the partitions, a feature vector is constructed containing the first two moments (mean and variance) and the joint moment (covariance) computed in the CIELAB color space. For each image in the collection, thus, a feature vector with 144 dimensions (i.e., 16 partitions times 3 moments times 3 color channels) is stored in *db*ADAM.

For the evaluation, we have used the ImageNet collection containing 14 million images [18]. The collection is organized according to the WordNet ontology. Hence, all images come with a synset identifier –and, thus, ultimately with keywords– that denotes a cognitive synonym to which an image belongs to.

We run ADAM on 28 small Ubuntu instances on Microsoft Windows Azure (1 core, 1.6 GHz CPU, 1.75 GB RAM). The full collection is sharded uniformly and randomly to the 28 instances, such that each shard stores about 500'000 images. The shards are coordinated and orchestrated by the *ws*ADAM component that runs as well from a machine within Windows Azure (4 cores, 1.6 GHz CPU, 7 GB RAM).

The extracted features in the *db*ADAM shards are all indexed using the equifrequent marks strategy. The process of creating an index over all features takes in the order of 2-3 minutes. The index uses about 27 MB disk space on 33'810 pages (each 8 KB) per shard making up a total of about 756 MB for the whole collection. It is evident that the size of the index largely depends on how many bits are assigned to each dimension. In our current implementation, each dimension is quantized using 64 marks and, thus, represented by 6 bits.

For the evaluation, we have gathered 66 hand-drawn sketches of 6 different images that are contained in the collection. These sketches are used as a basis to measure the systems' performance at a fixed collection size when using various query paradigms and options. Figure 7 displays the images used in the evaluation and an exemplary choice of the hand-drawn sketches.

We use the sketches as query input and search in the full collection of 14 million images while measuring the time to perform the retrieval. Since our focus is not on the extraction of features, which is highly dependent on the extraction algorithm used, we ignore the time to build up the query string and only consider the time for ADAM to retrieve the feature vector from the *db*ADAM shards and combine the results to a single set. To avoid anomalies in the results due to network congestions, etc., we have run each experimental setting 150 times (using the various sketches) and average over the retrieval times.

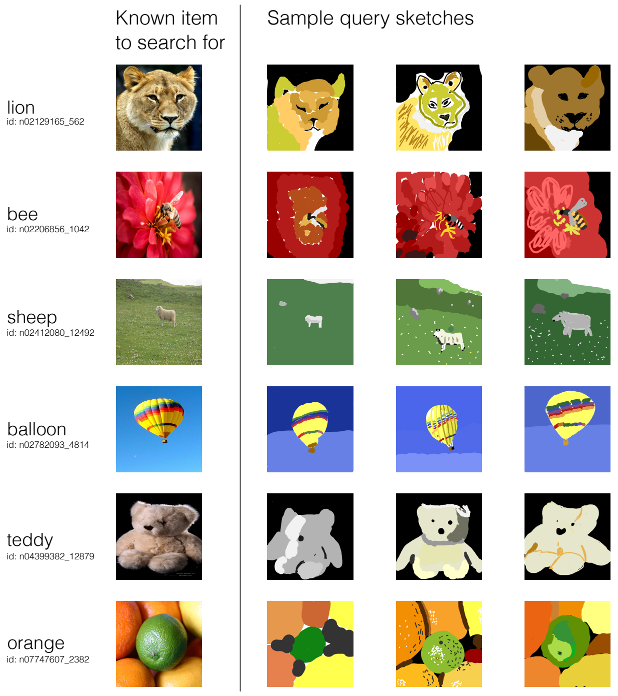


Figure 7. Exemplary sketches used in the evaluation of ADAM.

B. Retrieval Task

1) Evaluation of the shards: Table I displays the mean time of the single shards to answer a query under varying query parameters, e.g., when making use of VA indexing, when applying a keyword in the query, or when changing the number of results returned by each system (`LIMIT`) at a fixed collection size of 14 million images.

We display the same data in a box plot in Figure 8 (which we scale to the interval $[0, 5]$ for better readability of the query times near 0). Box plots are useful means to display the distribution of data. Figure 8 shows for the various sketches the distribution of query times over the different experimental runs and the different machines. The thick line within the box denotes the 50% quantile, i.e., the median, whereas the lower and the upper line of the box represent the 25% quantile and the 75% quantile, respectively. For the sketches that represent the "lion" image, for instance, 25% of the queries returned within 0.23s, in the median the reponse time was 0.41s and in 75% of the cases the queries were answered in 0.49s by a *db*ADAM shard. The dots represent outliers outside of the ± 1.5 interquartile range.

Table I and Figure 8 show that the use of the VA index structure heavily improves the retrieval time and the efficiency of answering a query. In all cases the mean (and the median, as the box plot shows) is lower when making use of VA indexing compared to when not using it. The table also shows that the size of the return set has no influence on the retrieval time. In contrast, the use of

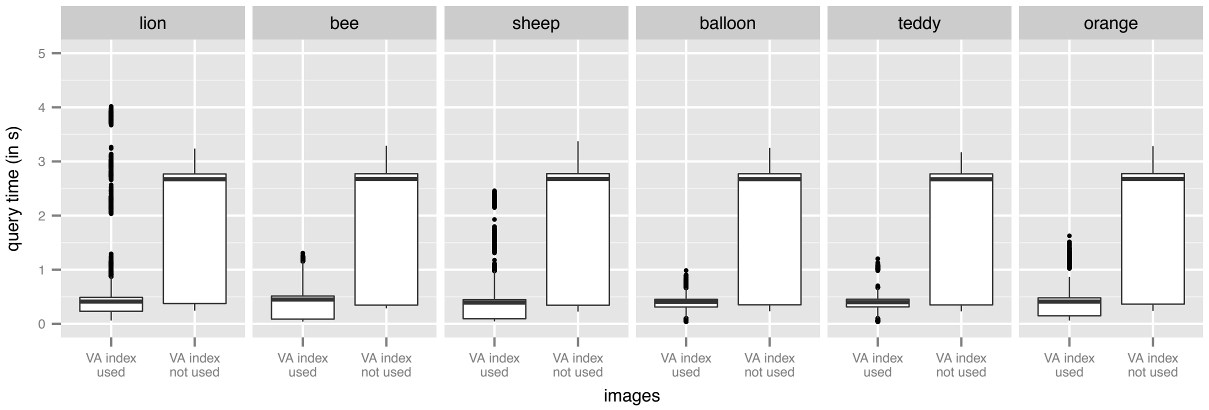


Figure 8. Box plot of the query time for the single shards scaled to the interval $[0, 5]$.

keywords surprisingly negatively affects the mean retrieval time when not using the index. We assume that this is due to many results adhering to the Boolean condition that finally lead to a large number of pages that need to be loaded separately. Thus, the combination of both index structures can only yield higher times, since it requires the loading of further database pages. Finally, when considering only the cases using the built-in index structures, very responsive query times can be achieved.

2) Evaluation of the ADAM system: So far we have only considered the query time of the single shards. In the following, we discuss the response time of the whole system. Since *ws*ADAM waits for all shards to answer a query (up to a fixed timeout of 150s) and only then returns the results to the query initiator, the response times of the whole ADAM system are higher than of the single shards. Table II shows the query time for the whole ADAM system.

Again, the table shows that the combination of Boolean retrieval and similarity retrieval compared to using similarity retrieval only is slower, even though very efficient B-tree index structures are exploited. In our evaluation using VA indexing, all similarity-based queries run in less than 0.7s; on the other hand, all compound queries, i.e., using both Boolean and similarity retrieval in combination, finished within 4s. Table II shows again that the use of the VA index structure largely improves retrieval. The median (not displayed in the table) of the query time when using VA lies at 0.66s, whereas when not making use of VA indexing the median is at 14.60s.

Overall, our evaluation has shown a very good performance in big multimedia collections, in particular when making use of the provided index structures. The behavior of ADAM with smaller and larger feature vector sizes has not yet been evaluated, as well as the number of bits per dimension (b_d) used for storing the quantized version of the feature vector. Furthermore, we have not yet evaluated the influence of the number of *db*ADAM shards and the behaviour of the retrieval system when increasing the number of parallel queries and the collection size, respectively. Due to the architecture of ADAM, however, we believe that it is possible to further increase the collection size by means of horizontal scaling without experiencing significant efficiency losses. Considering the mean response time of the total system for the current setting, we believe to present with ADAM a retrieval system that is able to handle big multimedia collections very efficiently.

Table I
QUERY TIME OF THE SHARDS IN SECONDS FOR EXPERIMENTS WITH DIFFERENT PARAMETERS FOR COLLECTION OF 14M IMAGES.

Index	Keywords	Limit	Mean Time
with VA	with kw	10	0.43 s
		50	0.41 s
		100	0.41 s
	without kw	10	0.42 s
		50	0.43 s
		100	0.46 s
without VA	with kw	10	9.5 s
		50	9.5 s
		100	9.5 s
	without kw	10	2.75 s
		50	2.75 s
		100	2.75 s

Table II
TOTAL QUERY TIME IN SECONDS FOR EXPERIMENTS WITH DIFFERENT PARAMETERS FOR COLLECTION OF 14M IMAGES.

Index	Keywords	Limit	Mean Time
with VA	with kw	10	1.76 s
		50	1.71 s
		100	1.76 s
	without kw	10	0.50 s
		50	0.52 s
		100	0.55 s
without VA	with kw	10	53.57 s
		50	53.86 s
		100	53.43 s
	without kw	10	3.00 s
		50	3.02 s
		100	3.01 s

VI. Conclusion

With ADAM, we have introduced a novel system that bridges the gap between databases and information retrieval systems for big multimedia data.

In terms of Big Data, ADAM tackles the aspect of variety by allowing to jointly store, manage and query structured and unstructured, heterogeneous data using various query paradigms, i.e., Boolean retrieval, similarity-based vector space retrieval and the combination of the two. While ADAM is evaluated using an image retrieval application, it is not bound to one specific retrieval problem or content type, but rather provides a generic solution for a large variety of search-based applications in the context of big multimedia retrieval. Since it leaves the implementation of the feature extraction to the user, while only considering the notion of a feature vector explicitly, the ADAM system is able to handle other multimedia data, as well. We are currently developing a video retrieval application that uses ADAM as the storage system for storing and retrieving feature vectors.

In terms of volume and velocity, ADAM has been designed and implemented to support efficiency and scalability of the retrieval process. First, in terms of the supported index structures, we have implemented vector approximation indexing within *db*ADAM to increase the performance of similarity retrieval in large collections. Second, from a systems point of view, ADAM uses *ws*ADAM for map and reduce operations in combination with the *db*ADAM shards to distribute the burden at retrieval time.

In our future work, we plan to evaluate ADAM using even larger collections and applying various –also more-compute intesive– feature extractions. We plan to further analyze possible ways of distributing data and to approach the problem of source selection in the ADAM system, which is a non-trivial task to solve if no Boolean predicates are available. Furthermore, we intend to test ADAM in a setting with a high number of parallel queries. Finally, we plan to extend our system with mechanisms to handle failures.

Acknowledgments

This work was partly supported by the Swiss National Science Foundation, project iMotion, and the Microsoft Windows Azure research grant, project ADAM$^+$.

References

[1] E. L. Wimmers and M. J. Carey, "Towards Heterogeneous Multimedia Information Systems: The Garlic Approach," in *Proc. 1995 Research Issues in Data Engineering*, Taipei, 1995, pp. 124–131.

[2] V. E. Ogle and M. C. Stonebraker, "Chabot: Retrieval from a Relational Database of Images," *Computer*, vol. 28, no. 9, pp. 40–48, 1995.

[3] A. P. de Vries, "Content and Multimedia Database Management Systems," Ph.D. dissertation, Centre for Telematics and Information Technology, Univ. of Twente, 1999.

[4] V. Oria, M. T. Oezsu, and P. Iglinski, "Querying Images in the DISIMA DBMS," in *Proc. 2001 Multimedia Information Systems*, Capri, 2001, pp. 89–98.

[5] C. E. Alvez and A. R. Vecchietti, "Combining Semantic and Content Based Image Retrieval in ORDBMS," in *Proc. 2010 Knowledge-based and Intelligent Information and Engineering Systems*, Cardiff, 2010, pp. 43–55.

[6] D. Moise et al., "Terabyte-scale Image Similarity Search: Experience and Best Practice," in *Proc. 2013 IEEE Big Data*, Santa Clara, 2013, pp. 674–682.

[7] A. Guttman, "R-Trees: A Dynamic Index Structure for Spatial Searching," in *Proc. 1984 ACM SIGMOD Int. Conf. Management of Data*, Boston, 1984, pp. 47–57.

[8] P. Ciaccia, M. Patella, and P. Zezula, "M-tree: An Efficient Access Method for Similarity Search in Metric Spaces," in *Proc. 1997 Very Large Data Bases*, Athens, 1997, pp. 426–435.

[9] S. Berchtold, D. A. Keim, and H.-P. Kriegel, "The X-Tree: An Index Structure for High-Dimensional Data," in *Proc. 1997 Very Large Data Bases*, Mumbai, 1997, pp. 28–39.

[10] R. Weber, H.-J. Schek, and S. Blott, "A Quantitative Analysis and Performance Study for Similarity-Search Methods in High-Dimensional Spaces," in *Proc. 1998 Very Large Data Bases*, New York, 1998, pp. 194–205.

[11] C. C. Aggarwal and P. S. Yu, "The iGrid Index: Reversing the Dimensionality Curse for Similarity Indexing in High Dimensional Space," in *Proc. 2000 ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining*, Boston, 2000, pp. 119–129.

[12] C. Calistru, C. Ribeiro, and G. David, "Multidimensional Descriptor Indexing: Exploring the BitMatrix," in *Proc. 2006 Int. Conf. Image and Video Retrieval*, Tempe, 2006, pp. 401–410.

[13] K. Böhm et al., "Fast Evaluation Techniques for Complex Similarity Queries," in *Proc. 2001 Very Large Data Bases*, Roma, pp. 211–220.

[14] D. G. Lowe, "Object Recognition from Local Scale-Invariant Features," in *Proc. 1999 IEEE Int. Conf. Computer Vision*, Kerkyra, 1999, pp. 1150–1157.

[15] W. Meng, C. Yu, and K.-L. Liu, "Building Efficient and Effective Metasearch Engines," *ACM Computing Surveys*, vol. 34, no. 1, pp. 48–89, 2002.

[16] J. Dean and S. Ghemawat, "MapReduce: A Flexible Data Processing Tool," *Communications of the ACM*, vol. 53, no. 1, pp. 72–77, Jan. 2010.

[17] A. Chalechale, A. Mertins, and G. Naghdy, "Edge Image Description using Angular Radial Partitioning," in *Vision, Image and Signal Processing*, vol. 151, no. 2, pp. 93–101, 2004.

[18] J. Deng et al., "ImageNet: A Large-Scale Hierarchical Image Database," in *Proc. 2009 IEEE Conf. Computer Vision and Pattern Recongition*, Miami, pp. 248–255.