Cottontail DB: An Open Source Database System for Multimedia Retrieval and Analysis

Ralph Gasser
ralph.gasser@unibas.ch
University of Basel, Switzerland

Luca Rossetto
rossetto@ifi.uzh.ch
University of Zurich, Switzerland

Silvan Heller
silvan.heller@unibas.ch
University of Basel, Switzerland

Heiko Schuldt
heiko.schuldt@unibas.ch
University of Basel, Switzerland

ABSTRACT

Multimedia retrieval and analysis are two important areas in "Big data" research. They have in common that they work with feature vectors as proxies for the media objects themselves. Together with metadata such as textual descriptions or numbers, these vectors describe a media object in its entirety, and must therefore be considered jointly for both storage and retrieval.

In this paper we introduce Cottontail DB, an open source database management system that integrates support for scalar and vector attributes in a unified data and query model that allows for both Boolean retrieval and nearest neighbour search. We demonstrate that Cottontail DB scales well to large collection sizes and vector dimensions and provide insights into how it proved to be a valuable tool in various use cases ranging from the analysis of MRI data to realizing retrieval solutions in the cultural heritage domain.

CCS CONCEPTS

• **Information systems** → *Data management systems*; **Multimedia databases**; **Multimedia and multimodal retrieval**.

KEYWORDS

Multimedia Retrieval, Multimedia Indexing, Database, Open Source

ACM Reference Format:
Ralph Gasser, Luca Rossetto, Silvan Heller, and Heiko Schuldt. 2020. Cottontail DB: An Open Source Database System for Multimedia Retrieval and Analysis. In *Proceedings of the 28th ACM International Conference on Multimedia (MM '20), October 12–16, 2020, Seattle, WA, USA.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3394171.3414538

1 INTRODUCTION

Multimedia data is ubiquitous and can be found in countless applications across many different domains ranging from medical research to entertainment and fraud detection. Such multimedia data usually comes in huge quantities and in different forms, including but not limited to images, audio, videos, data streams, etc. This variety and the sheer volume make *multimedia analysis* and *multimedia retrieval* two core disciplines of the more general domain of "Big data" research, with very specific requirements on data storage, access and processing.

In addition to the original multimedia objects and the associated metadata, a lot of analysis and retrieval techniques rely on more compact representations which are stored and processed alongside the aforementioned information. These representations, often referred to as *features*, can encode very simple properties, e.g., the average color of an image, but also very complex information, e.g., the output of a hidden layer of an artificial neural network. What these features have in common, however, is that mathematically they can be seen as d-dimensional vectors in $\mathbb{N}^d$, $\mathbb{R}^d$ or even $\mathbb{C}^d$. Downstream applications then operate on an ensemble of these *feature vectors*, e.g., for classification, statistics, or lookup.

In the *vector space model* of multimedia retrieval, for example, feature vectors belonging to a specific item in a collection are seen as points in a d-dimensional vector space. The dissimilarity between two objects is then expressed as the distance D between two such vectors. Very often, simple distance metrics such as the Minkovski-distances (L_1, L_2) are employed. However, a great number of other distance measures exist [7], depending on the use case.

If one now considers metadata and feature vectors jointly, it becomes apparent that multimedia analysis and retrieval poses very distinct requirements on a potential database engine: Firstly, such an engine should be able to store feature vectors, textual information, numbers, etc. in a unified data model. Secondly, its users should be enabled to express different types of queries such as classical Boolean queries (e.g., "Find all images, that have been captured by a specific person.") and similarity queries (e.g., "Find all video segments, whose feature vectors are close to the given example.") as well as a combination thereof using one query language. And finally, it should be able to cope with large volumes of data, especially since feature vectors nowadays can become very high-dimensional while at the same time, large quantities of these vectors are produced in multimedia analysis scenarios.

In this paper we introduce *Cottontail DB*, an open source[1] database management system that jointly supports classical Boolean retrieval as well as vector space retrieval. Cottontail DB is a column store implemented in Kotlin and it is based on the concepts and ideas introduced in [5]. Today, Cottontail DB is an integral part of the *vitrivr* stack [4, 16] and has been battle-tested on several

MM '20, October 12–16, 2020, Seattle, WA, USA

ACM ISBN 978-1-4503-7988-5/20/10…$15.00
https://doi.org/10.1145/3394171.3414538

[1] https://github.com/vitrivr/cottontaildb

occasions. Among others, it has successfully been used to power vitrivr's participation to interactive retrieval competitions such as the Video Browser Showdown (VBS) [14] and the Lifelog Search Challenge (LSC) [6], contributing substantially to vitrivr achieving high scores in both evaluation campaigns in their most recent editions [13, 17]. Furthermore, it forms the basis for a multitude of multimedia retrieval and analysis activities, ranging from the cultural heritage domain to signal analysis in magnetic resonance fingerprinting [12]. As such, Cottontail DB has proven to be a versatile and powerful tool when it comes to managing, accessing, and processing data related to multimedia collections.

The contribution of this paper is twofold: First, we introduce Cottontail DB and provide insights into its system architecture. Second, we compare Cottontail DB to ADAM$_{pro}$ [5] and demonstrate how it scales to large media collections containing millions of items.

The remainder of this paper is structured as follows: Section 2 surveys related work and Section 3 provides an overview over Cottontail DB's system architecture. Section 4 compares the retrieval performance of Cottontail DB to its predecessor ADAM$_{pro}$ and provides some insights into how it scales to even larger datasets. Section 5 concludes and provides an outlook to future work.

2 RELATED WORK

Over the years, the need for having efficient solutions that can provide both nearest neighbour lookup for similarity search and classical Boolean retrieval has brought about many different systems and tools. Early examples of such systems can be tracked as far back as the 1980s [18] and 1990s [2, 3]. A lot of the available solutions, however, focus on either one of the two aspects. There are plenty of examples for classical DBMS solutions, with many free and open source systems such as PostgreSQL[2] or Maria DB[3] available. One example from the multimedia retrieval domain is the Lucene Image Retrieval (LIRE) [11] system. LIRE is a content based image retrieval library that leverages *Apache Lucene*[4] for indexing and storage and comes with a plugin for *Apache Solr*. It comes bundled with a great number of local and global features, such as MPEG-7 descriptors. However, it has not been designed as a general purpose storage solution for multimedia data and is limited for use with the features and distance metrics it comes with. FAISS [10] developed by Facebook research is another example of such an open source library. FAISS claims to facilitate similarity search and clustering on billion-scale datasets by leveraging the computational power of GPUs. But FAISS does not handle persistent storage and only operates on data structures in RAM. Furthermore, its platform compatibility is limited since there are only bindings for C++ and Python and GPU acceleration only works with NVIDIA GPUs.

More in line with the idea of a unified architecture for multimedia and Boolean retrieval is ADAM$_{pro}$ [5], from which Cottontail DB took some inspiration in terms of concepts and ideas. The idea behind ADAM$_{pro}$ is to have a unified data model and query language that allows to query collections consisting of primitive data types (i.e., numbers, dates, strings, etc.) and vectors. It does so by combining different storage engines and having an orchestration engine on top that facilitates query planning and execution. In addition, it comes with a selection of exact and approximate indexing techniques that speed-up nearest neighbour search, including, but not limited to, *VA-File* [21], *Locality Sensitive Hashing* (LSH) [8] and *Product Quantization* (PQ) [9] indexes. However, even though ADAM$_{pro}$ was shown to scale well to collection sizes in the range of tens of millions of items and beyond [5], due to some internal design decisions, it came with a significant overhead, which was particularly detrimental for smaller collections and led to relatively long retrieval times in such scenarios.

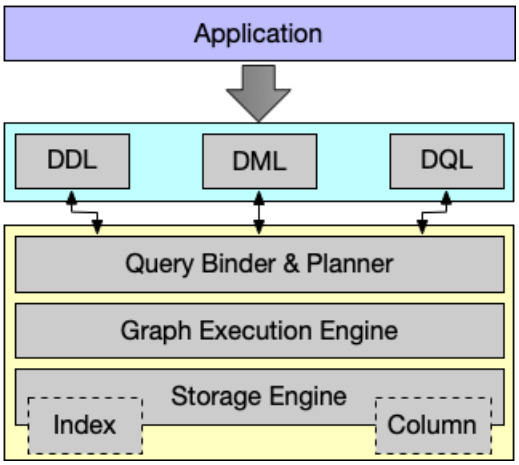

Figure 1: Illustration of Cottontail DB's system architecture.

3 COTTONTAIL DB: SYSTEM OVERVIEW

3.1 Architecture

Cottontail DB does away with the idea of having different storage engines for different data types and instead uses the same storage engine for all data types it supports. Conceptually, Cottontail DB is a column store, that is, data belonging to the same column is stored in the same file. Columns that semantically belong together are grouped in *entities*, which is the equivalent to a table in a classical DBMS. Each record in such an entity is identified by a *tuple ID*. Furthermore, entities can be organized into different *schemata* and Cottontail DB can manage many schemata in a single instance.

Each column in Cottontail DB is strongly typed. Here we leverage a type system that maps to primitive data types of the underlying Java Virtual Machine (JVM). In addition to supporting primitive data types such as strings, integer numbers, and floating point numbers, Cottontail DB allows to compose these primitive types into structured types such a *float vectors*, *complex numbers*, or *vectors of complex numbers*. Support for the latter has been added to facilitate research in MRI signal analysis. Mathematical operations such as addition, subtraction, multiplication, and division are defined on primitive numerical types as well as on vector types. On top of these basic operations, Cottontail DB implements various distance kernel functions required for nearest neighbour search. Currently, it supports the Manhattan (L_1) distance, the Euclidean (L_2) distance, the Cosine distance, the Hamming distance and the Chi-Squared distance. However, other distance kernels can be added very easily.

[2]https://www.postgresql.org/
[3]https://mariadb.org/
[4]https://lucene.apache.org/

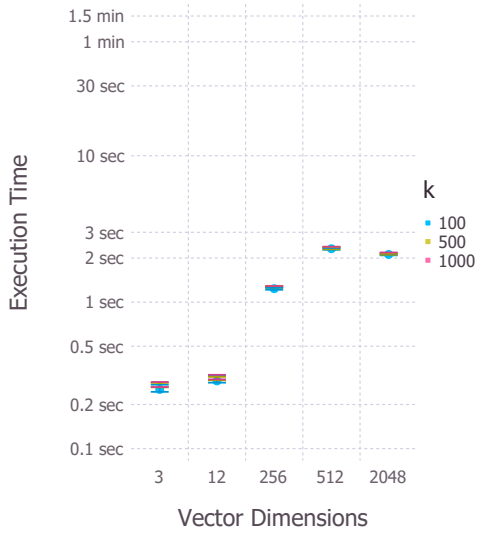

Figure 2: Query execution times on Cottontail DB.

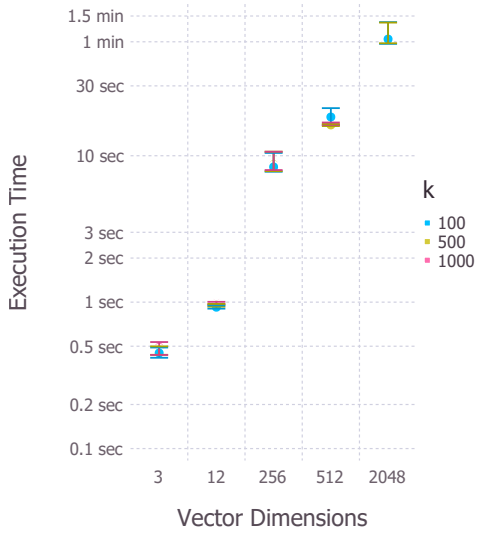

Figure 3: Query execution times on ADAM$_{pro}$.

An overview of Cottontail DB's system architecture is given in Figure 1. An application can interact with Cottontail DB through a defined programming interface, as it would with any other database. The interface exposes all relevant functionality such as the definition of schemata and entities (DDL), insertion and management of data (DML), and querying (DQL). On a very high level, requests formulated through the exposed API are first handed to an engine that binds names to objects in Cottontail DB's data model and then generates an execution plan for the query. This execution plan is then handed to a graph-based execution engine, which interacts with the required schemata, entities, columns, and indexes and generates the desired result set. To speed-up retrieval time, Cottontail DB offers support for various index structures both for nearest neighbor search (e.g., LSH-based indexes) and Boolean search (e.g., hash-based indexes) and allows for trade-offs between retrieval speed and accuracy through the use of inexact index structures.

3.2 Design Considerations

Cottontail DB has been designed with modularity and extensibility in mind. Many different aspects of Cottontail DB can be extended simply by implementing a certain interface. For example, it is straightforward to add new index structures for nearest neighbour or Boolean search or new types of kernel functions for distance calculation. Even the interaction with the low-level storage engine is defined via a set of interfaces and thus, the engine can be replaced easily. This has proven valuable when adjusting Cottontail DB to a new use case and extending it with new functionality. For example, integrating and experimenting with different indexing techniques for nearest neighbor search in MRI signal analysis proved to be very simple given Cottontail DB's versatile architecture.

Some of the design considerations, however, run very deep in the code base. For example, Cottontail DB being a column store allows for several optimizations on different levels especially in the domain of nearest neighbour search. First and foremost, only the data relevant to the query is actually read from disk so as to minimize I/O during such a search operation. Furthermore, the compact organization of data allows for parallelisation. The search space can simply be partitioned into smaller chunks and each chunk can then be handed to a dedicated thread for processing.

3.3 Implementation

Cottontail DB is written in Kotlin[5] and runs on the JVM. Hence, it can be used on many different platforms. The API that exposes all the functionality Cottontain DB offers is specified in gRPC[6], which allows the use of Cottontail DB with a huge number of different programming languages and platforms.

Currently, Cottontail DB uses a fork of Map DB[7] for data storage. Map DB is a library that provides fast and persistent data structures such as maps, queues, and lists. Furthermore, it comes with an API for low-level data organisation. One advantage of Map DB is that it uses memory mapped files and *mmap*, which facilitates very fast access to the underlying data. A custom storage engine without the reliance on *mmap* that allows a more fine grained control over I/O organization is currently in development.

4 EVALUATION

To demonstrate Cottontail DB's ability to cope with large multimedia collections, we compare ADAM$_{pro}$ and Cottontail DB in terms of retrieval speed for nearest neighbour queries. We base the evaluation on the V3C1 [1] video collection, which has an overall collection size of $s = 1.088.896$ segments and is used in the VBS setting and therefore representative of interactive video retrieval tasks. For the comparison, we generated features using Cineast [15] with vector dimensions ranging from $d_{min} = 3$ to $d_{max} = 2048$ and compared them using the Euclidean (L2) distance. All tests

[5] https://kotlinlang.org/
[6] https://grpc.io/
[7] http://www.mapdb.org/

were run on the same hardware (32 cores at 3.4 GHz, 64 GB RAM, data resides on SSDs) and both Cottontail DB and ADAM$_{pro}$ were allowed to use 8 GB of the available memory. All measurements were aggregated over ten runs. Before the actual runs started, three warm-up queries were conducted to initialize caches and other data structures. Figures 2 and 3 depict the query execution times for linear scans without the additional use of any index structure for Cottontail DB and ADAM$_{pro}$, respectively, grouped by feature dimension and number of returned elements (k). Note that we are showing a logarithmic time scale so as to enable better side by side comparison. It can be seen that Cottontail DB not only consistently outperforms ADAM$_{pro}$ in terms of execution time, it also exhibits a smaller variance. The dip in execution time for $d = 2048$ compared to $d = 512$ can be attributed to an internal parallelization heuristic in Cottontail DB, which apparently achieves a higher resource utilization in the case of the longer feature vector.

To demonstrate that Cottontail DB can cope with even larger collections, we conducted a similar experiment on the YFCC100M [19] dataset, which contains 100 million images, i.e., roughly a 100 times more elements than V3C1. In contrast to our comparison with ADAM$_{pro}$, we used pre-generated features from the multimedia commons[8] dataset, more precisely, feature vectors generated by LIRE. In our experiment, nearest neighbour lookup execution times ranged between 70.69s ± 2.81s for *auto color correlogram* features (acc, $d = 64$) and 500.42s ± 4.90s for *joint histogram* features (jhist, $d = 576$) without the use of any index. By using an LSH index with a 100 buckets, execution time could be brought down to 0.87s ± 0.28s and 3.60s ± 0.041s, respectively.

5 CONCLUSION AND FUTURE WORK

In this paper we introduced Cottontail DB – our open source database management system for multimedia collections – and demonstrated its capability to manage and query large multimedia collections. Cottontail DB combines Boolean and nearest neighbour based retrieval and can be used to perform all types of analyses on multimedia collections. Its well-defined programming interfaces and its open architecture make it a great foundation for all kinds of applications in that domain.

Currently, we are working on a new low-level storage engine for Cottontail DB, which will enable more fine grained control over I/O and buffering of data pages while at the same time taking advantage of certain access patterns inherent to certain types of queries and data. We also plan to add support for SQL by integrating Cottontail DB into the polystore *Polypheny DB* [20]. Long-term ideas involve support for distribution to several nodes and leveraging SIMD instructions for nearest neighbour search operations through Java's project *Panama*[9] and JVM support for vector intrinsics.

Even though Cottontail DB may still be a very young project, it has already propelled some of our own activities forward. We have seen the advantage of having such a scalable, modular and extensible database system and we strongly believe, that its versatility will prove valuable to other multimedia researchers as well as concrete applications in the industry.

[8] http://mmcommons.org/
[9] https://openjdk.java.net/projects/panama/

REFERENCES

[1] Fabian Berns, Luca Rossetto, Klaus Schoeffmann, Christian Beecks, and George Awad. 2019. V3C1 Dataset: An Evaluation of Content Characteristics. In *Proceedings of the 2019 ACM International Conference on Multimedia Retrieval* (Ottawa ON, Canada) *(ICMR '19)*. Association for Computing Machinery, New York, NY, USA, 334–338. https://doi.org/10.1145/3323873.3325051

[2] Michael J Carey, Laura M Haas, Peter M Schwarz, Manish Arya, William F Cody, Ronald Fagin, Myron Flickner, Allen W Luniewski, Wayne Niblack, Dragutin Petkovic, et al. 1995. Towards heterogeneous multimedia information systems: The Garlic approach. In *Proceedings RIDE-DOM'95. Fifth International Workshop on Research Issues in Data Engineering-Distributed Object Management*. IEEE, Taipei, Taiwan, 124–131.

[3] Arjen P de Vries and HM Blanken. 1998. Database technology and the management of multimedia data in the Mirror project. In *Multimedia Storage and Archiving Systems III*, Vol. 3527. International Society for Optics and Photonics, Boston, MA, USA, 443–453.

[4] Ralph Gasser, Luca Rossetto, and Heiko Schuldt. 2019. Multimodal multimedia retrieval with Vitrivr. In *Proceedings of the 2019 ACM International Conference on Multimedia Retrieval*. ACM, Ottawa, ON, Canada, 391–394.

[5] Ivan Giangreco and Heiko Schuldt. 2016. ADAM$_{pro}$: Database support for big multimedia retrieval. *Datenbank-Spektrum* 16, 1 (2016), 17–26.

[6] Cathal Gurrin, Klaus Schoeffmann, Hideo Joho, Andreas Leibetseder, Liting Zhou, Aaron Duane, Duc-Tien Dang-Nguyen, Michael Riegler, Luca Piras, Minh-Triet Tran, et al. 2019. [Invited papers] Comparing Approaches to Interactive Lifelog Search at the Lifelog Search Challenge (LSC2018). *ITE Transactions on Media Technology and Applications* 7, 2 (2019), 46–59.

[7] Rui Hu, Stefan Ruger, Dawei Song, Haiming Liu, and Zi Huang. 2008. Dissimilarity measures for content-based image retrieval. In *IEEE International Conference on Multimedia and Expo*. IEEE, Hannover, Germany, 1365–1368.

[8] Piotr Indyk and Rajeev Motwani. 1998. Approximate nearest neighbors: towards removing the curse of dimensionality. In *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing*. ACM, Dallas, TX, USA, 604–613.

[9] Herve Jegou, Matthijs Douze, and Cordelia Schmid. 2010. Product quantization for nearest neighbor search. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 33, 1 (2010), 117–128.

[10] Jeff Johnson, Matthijs Douze, and Hervé Jégou. 2017. Billion-scale similarity search with GPUs. *CoRR* abs/1702.08734 (2017).

[11] Mathias Lux. 2011. Content Based Image Retrieval with LIRe. In *Proceedings of the 19th ACM International Conference on Multimedia* (Scottsdale, Arizona, USA) *(MM '11)*. Association for Computing Machinery, New York, NY, USA, 735–738. https://doi.org/10.1145/2072298.2072432

[12] Dan Ma, Vikas Gulani, Nicole Seiberlich, Kecheng Liu, Jeffrey L Sunshine, Jeffrey L Duerk, and Mark A Griswold. 2013. Magnetic resonance fingerprinting. *Nature* 495, 7440 (2013), 187–192.

[13] Luca Rossetto, Ralph Gasser, Silvan Heller, Mahnaz Amiri Parian, and Heiko Schuldt. 2019. Retrieval of structured and unstructured data with vitrivr. In *Proceedings of the ACM Workshop on Lifelog Search Challenge*. ACM, Ottawa, ON, Canada, 27–31.

[14] Luca Rossetto, Ralph Gasser, Jakub Lokoc, Werner Bailer, Klaus Schoeffmann, Bernd Muenzer, Tomas Soucek, Phuong Anh Nguyen, Paolo Bolettieri, Andreas Leibetseder, et al. 2020. Interactive Video Retrieval in the Age of Deep Learning-Detailed Evaluation of VBS 2019. *IEEE Transactions on Multimedia* (2020).

[15] Luca Rossetto, Ivan Giangreco, and Heiko Schuldt. 2014. Cineast: a multi-feature sketch-based video retrieval engine. In *2014 IEEE International Symposium on Multimedia*. IEEE, Taichung, Taiwan, 18–23.

[16] Luca Rossetto, Ivan Giangreco, Claudiu Tanase, and Heiko Schuldt. 2016. vitrivr: A flexible retrieval stack supporting multiple query modes for searching in multimedia collections. In *Proceedings of the 24th ACM international Conference on Multimedia*. ACM, Amsterdam, The Netherlands, 1183–1186.

[17] Loris Sauter, Mahnaz Amiri Parian, Ralph Gasser, Silvan Heller, Luca Rossetto, and Heiko Schuldt. 2020. Combining Boolean and Multimedia Retrieval in vitrivr for Large-Scale Video Search. In *International Conference on Multimedia Modeling*. Springer, 760–765.

[18] Hans-Jörg Schek. 1980. Methods for the administration of textual data in database systems. In *Proceedings of the 3rd annual ACM Conference on Research and Development in Information Retrieval*. Butterworth & Co., Cambridge, UK, 218–235.

[19] Bart Thomee, David A. Shamma, Gerald Friedland, Benjamin Elizalde, Karl Ni, Douglas Poland, Damian Borth, and Li-Jia Li. 2016. YFCC100M: The New Data in Multimedia Research. *Commun. ACM* 59, 2 (Jan. 2016), 64–73. https://doi.org/10.1145/2812802

[20] Marco Vogt, Alexander Stiemer, and Heiko Schuldt. 2018. Polypheny-DB: Towards a Distributed and Self-Adaptive Polystore. In *2018 IEEE International Conference on Big Data (Big Data)*. IEEE, Seattle, WA, USA, 3364–3373.

[21] Roger Weber, Hans-Jörg Schek, and Stephen Blott. 1998. A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces. In *VLDB*, Vol. 98. Morgan Kaufmann, New York City, NY, USA, 194–205.