INDEXING OF MPEG-7 STREAMS[1]

Andrea Kofler-Vogt
University Klagenfurt
Austria
andrea@itec.uni-klu.ac.at

Harald Kosch
University Klagenfurt
Austria
harald@itec.uni-klu.ac.at

Jörg Heuer
Siemens AG; Munich
Germany
Joerg.Heuer@siemens.com

ABSTRACT

The ISO/IEC Motion Picture Group (MPEG) issued in 2002 a standard, called MPEG-7, which enables the content description of multimedia data in XML. The standard supports applications to exchange, identify and filter multimedia contents based on MPEG-7 descriptions. However, processing MPEG-7 documents on mobile terminals is problematic, since the verbose XML is not adequate to limited bandwidth, low computational power and limited battery life. In this document we describe an index system that allows filtering and random access to encoded MPEG-7 streams and which overcomes the limitation of the network and the consuming terminal. Encoding is applied in order to reduce the data rate of the XML documents to be transmitted. The indexed parts of the encoded streams can be accessed without the need to deserialize the complete stream. Furthermore, the system is evaluated and results of the experimental evaluation are discussed.

1. INTRODUCTION

MPEG-7 ([1], [2], [3]) is a multimedia description standard which has been standardized by the ISO/IEC Motion Picture Expert Group (MPEG) to enable content providers and consumers to identify and search multimedia by content. Since its introduction in 2002, an increasing number of applications uses MPEG-7 to exchange their multimedia data. Application scenarios are described in [4], [2], and [5].

Since especially in wireless communication bandwidth is extremely expensive respectively limited, a compression mechanism for XML based MPEG-7 data is required. Due to this, the deserialization of compressed XML data must be efficient with respect to computational complexity. The compression mechanism has not only to be efficient in terms of the reduction of the size and processing but should also work in a streaming environment and should allow the encoding and filtering of independent parts of the XML document.

In this context, the system part of MPEG-7 introduced a efficient representation of XML called Binary Format of MPEG-7 (BiM) [6] to enable efficient transmission of the XML based MPEG-7 data. In this representation, the meta data is sent in a number of access units (AUs), each containing several fragment update units (FUUs). Accordingly by fragmentability a prerequisite of random access is fulfilled. To support random access on dedicated information within the stream index information for BiM-encoded meta data is required. In this paper we present a format for index information which can be transmitted alongside with the XML data and allows fast random access to particular fragment update units within an encoded stream. Additionally we describe the processing of the index and present measurements for the evaluation of the system.

2. RELATED WORK

XML Compression Tools. A couple of compression tools were developed that take advantage of the structure of an XML document. Examples for such tools are XMill [7], Millau [8], XGrind [9], and XPRESS [10]. In the case of MPEG-7, a compression tool was required which achieves a high compression rate, which can be used in a streaming environment, and which allows the compression of independent sub-trees. In order to provide these functionalities, MPEG-7 standardized its own binary format called BiM [6].

However, random access mechanisms are not provided by the first version of BiM. Instead, the complete stream has to be parsed before a query can be processed. For fast random access to desired FUUs, an index system is required.

XML Indexing Strategies. A number of indexing solutions for XML have been proposed; among them are the Index Fabric [11], SphinX [12], a multi-dimensional indexing strategy [13], XISS [14], APEX [15], and ToXin ([16], [17]).

Most proposed systems are optimized according to extended query functionality and provide for this a

[1] This project was funded in part by FWF (Fonds zur Förderung der wissenschaftlichen Forschung) P14789 and by KWF (Kärntner Wirtschaftsförderungsfonds).

complex set of index structures. Mapping these kinds of indexes into one stream is difficult, first due to the complex reference mechanisms when index trees are aligned. Second, these index structures are often not optimized for the requirement of a small storage size, which is crucial in the streaming context described in this paper. Thus, we will introduce an index system that is based on a B-Tree.

3. REQUIREMENTS

The index system we are looking for has to fulfill the following requirements:

- The data in the application scenarios we are considered is continuously streamed and re-streamed. Thus, our index system has to work in a streaming environment.
- The index system is applied in environments with limited resource availability (e.g., bandwidth). Due to this, the size of the index data to be sent is a crucial factor.
- The end devices used in many applications often suffer from limited processing power. Anyway, the index look-up in a streaming environment has to be performed in a small and well-defined time period.

4. THE IMPLEMENTATION OF THE INDEX SYSTEM

4.1. THE UNDERLYING DATA STRUCTURE

Different data structures were examined at the beginning.. According to the requirements listed above, we considered only data structures that provide logarithmic search behaviour, among them the sorted array, the binary tree, and the B-Tree (**Fehler! Verweisquelle konnte nicht gefunden werden.**, [18]). After some evaluations and estimations we decided to choose the B-Tree as basic data structure due to following reasons:

- The B-Tree is a field-proofed data structure for fast index-access in databases and file repositories.
- It supports the block-oriented partition of the memory in mobile devices, i.e., it is quite expensive to load a new block in mobile devices and the block size is quite small. The B-tree has been proven to be the most efficient data-structure in such memory layout, minimizing the number of accessed data blocks.
- Efficient search behavior. Since a B-tree is balanced, the logarithmic search behavior can be guaranteed.
- The B-tree may be applied to any kind of data.
- Updates caused by insertions or deletions affect only a limited number of nodes

Once this decision was done, we had to think about a mechanism that allows the usage of the B-Tree in the streaming environment. First of all, we had to find a way to index XML based MPEG-7 data with a B-Tree. We decided to insert root-to-leaf paths of the document tree in the index tree. Afterwards, we developed an efficient coding schema of the B-tree (streamlined B-tree) to optimize the space consumption according to the network bandwidth restrictions.

4.2. THE USAGE OF A B-TREE TO INDEX XML-DATA

In order to index MPEG-7 data with a B-Tree, first the relevant paths of a document to be indexed are generated. These paths, without the context and the values of attributes, serve as keys. They are inserted into the index tree.

Fig. 1 shows parts of the generated B-Tree, after all root-to-leaf path of a small example document were inserted. The same path may occur in a document several times with different instance values. However, a key is added only once into the tree. For this reason, the following information is transmitted for each key additionally: the number of occurrences of the path in the document tree, the values of attributes or the context of elements, and the location(s) of the FUU(s) containing the indexed nodes.

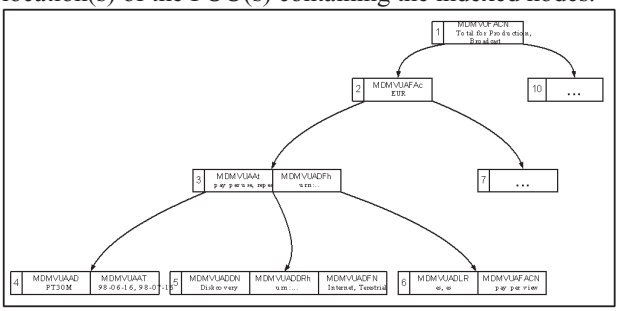

Fig. 1: Parts of a B-Tree

4.3. THE MAPPING OF THE TREE TO A LINEAR STREAM

Once the index structure is generated, the information is stored in the index stream. In a typical scenario the index stream is sent together with or before the description stream. In any case, the hierarchical B-Tree is mapped to a linear stream.

For the stream mapping, the tree is traversed in a depth-first manner, and the nodes are packed into the stream in the order they are visited. This order is signaled in Fig. 1 with the node numbers of the tree.

For each node in the stream, the start offset of all children beside the first child is coded too. This enables a skipping of undesired node information. If a node does not have any children, the offset is set to zero. For the first child

node, the offset is not transmitted since it is coded immediately after its parent node. Fig. 2 shows parts of the stream for the B-Tree represented above.

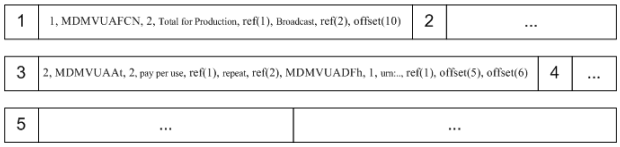

Fig. 2: Example of an Index Stream

4.4. THE INDEX LOOK-UP BASED ON THE TREE STREAM

For a look-up we consider an XPath [20] like query string as input. An index look-up in the B-Tree starts at the root node. Through comparison of the search pattern with the keys it is decided in which child node the searched entry can be found. Thus, if the desired key is not in the parent node only one child has to be visited.

The usage of offsets in the index stream enables one to directly process the proper node without having to parse the information of other nodes stored in between both nodes in the stream. Thus, during the look up, particular node information are read until the node content matches the query string, or no more children are available. In the latter case, the desired information is not present in the indexed document.

As an example, the following query: 'Mpeg7/Description/MultimediaContent/Video/Usage Information/Availability/Dissemination/Disseminator/Agent/Name' where the name of the agent is "Discovery" (short MDMVUADDAN = "Discovery") should be processed. Comparing the query string with the entry in the first node leads to a further proceeding in the second node. Then the search proceeds at the node with the number 5. This node can be directly accessed through the usage of the coded child offset. Here, the string matches with the first key and the FUU containing the desired element can be identified. The search path through the original B-Tree for this example is presented in Fig. 3.

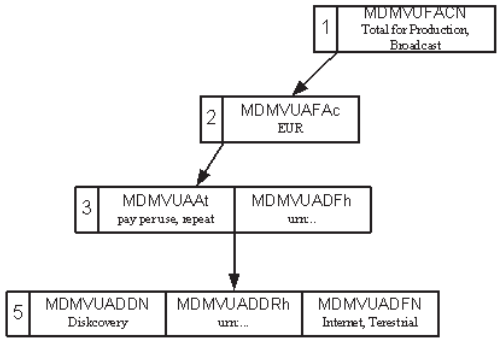

Fig. 3: An Example of a Search Path

In this example, only the information of the nodes 1, 2, 3, and 5 are parsed, the other data in the stream is skipped or ignored.

5. IMPROVEMENTS

5.1. COMPACTER CODING OF THE PATHS

First experiments have shown that the size of the index information may become almost that big as the size of the BiM encoded data. Thus, we considered improvements in order to reduce the index size. We examined several possibilities like the compression of the string repository using an entropy codec, coding of the XPaths relative to their neighbor or parent paths, or the binary coding of the paths as proposed in the MPEG-7 standard [6]. We implemented the binary coding of the key paths. A binary encoded path consists of a sequence of tree branch codes (TBCs) which are tokens for the elements appearing in this path followed by a sequence of position codes for those elements that may occur more than once. Since the positions do not influence the search in the index tree they were omitted. Experiments have shown that the compression of the paths can lead to a reduction of the index size of about 20%.

5.2. EXTENDED QUERY FUNCTIONALITY

Because our prototype was optimized concerning the storage size the position codes were omitted in our first approach. But while they do not influence the navigation within the B-Tree we propose to use them to answer more complex queries. In a modified version of our index system still only the tree branch codes serve as key entries in the B-Tree. But in order to provide a more complex query functionality the position codes are transmitted too.

In [21] we have shown, how position codes can be used to answer multiple-field queries, i.e. queries that contains at least to conditions.

5.3. DATA ORGANIZATION

While the indexed paths were stored in a tree, the instances belonging to a certain paths were not sorted. Due to this, a linear search of all values belonging to one path was required. To overcome the this problem, we decided to implement a data organization that allows a more efficient search within the indexed values. As first approach we built a data structure called value tree for each key entry. A detailed explanation of this concept can be found in [21].

6. CONCLUSION AND OUTLOOK

In this work, we proposed an index system that provides fast random access to binary encoded MPEG-7 descriptions. The approach was motivated by the increased usage of MPEG-7 in mobile multimedia application relying on metadata for identification, filter and search, for instance Electronic Program Guides (EPG), Multimedia Mail Services, etc.

With our approach, we have shown that the B-Tree is a well suited data structure to index XML-based MPEG-7 data. We proposed a coding scheme that allows the streaming of an index tree. Additionally, we improved our prototype by applying the binary coding of XML based path structures in order to reduce the index size. The functionality was extended to support multiple-field queries. For this purpose, we utilize the position codes of the MPEG-7 BiM. Finally, we achieved a decrease in look-up time by the usage of value trees for the element or attribute values which belong to the same path.

Future work will address further improvements on the index size. Especially compact e.g. differential encoding of position codes have to be investigated. Based on the observation that on average 35% of the present value stream consists of position information improvement in the position coding can have significant effect on the index size. With respect to processing efficiency we will evaluate the gain of the reordering of entries in the index tree, when the frequency of different queries is considered too. At the moment, we assume a uniform distribution of the key entries in the tree.

9. REFERENCES

[1] Jos M. Martinez. “Overview of the MPEG-7 Standard”, N4980, Klagenfurt, July 2002. Available from http://www.chiariglione.org/mpeg/standards/mpeg-7/mpeg-7.htm.

[2] B.S. Manjunath, P. Salembier and T. Sikora. “Introduction to MPEG-7.” John Wiley & Sons Ltd., 2002.

[3] J. Heuer, J.L. Casas, A. Kaup: “Adaptive Multimedia Messaging Based on MPEG-7 - The M3-Box“.Proc. Second International Symposium on Mobile Multimedia Systems & Applications, pp. 6-13, Delft, Nov. 2000

[4] Harald Kosch: “Distributed Multimedia Database Technologies supported by MPEG-7 and MPEG-21”, CRC Press, 248 pages, November 2003

[5] Official homepage of the TV-Anytime forum (see http://www.tv-anytime.org/).

[6] U. Niedermeier, J. Heuer, A. Hutter, W. Stechele, A. Kaup. ”An MPEG-7 Tool for Compression and Streaming of XML Data”, Proc. IEEE International Conference on Multimedia and Expo, pages 521-524, Lausanne, Switzerland, August, 2002.

[7] H. Liefke and D. Suciu. ”XMill: An Efficient Compressor for XML Data”. Proc. of the ACM SIGMOD International Conference on Management of Data, pages 153-164, Dallas, Texas, USA, May 2000.

[8] M. Girardot and N. Sundaresan. ”Millau: An Encoding Format for Efficient Representation and Exchange of XML over the Web”. 9th International World Wide Web Conference, www9.org, Amsterdam, The Netherlands, May 2000.

[9] P. Tolani and J.R. Haritsa. “XGRIND: A Query-friendly XML Compressor”. Proc. of IEEE Int. Conf. on Data Engineering (ICDE 2002), pages 225-235, San Jose California, USA, February-March 2002.

[10] J. K. Min, M. J. Park and C. W. Chung, ”XPRESS: A Queriable Compression for XML Data,”. Proc. of the ACM SIGMOD International Conference on Management of Data, San Diego

[11] B. Cooper, N. Sample, M. Franklin, G. Hjaltason and M. Shadmon. “A Fast Index for Semistructured Data”. Proc. of the VLDB Conference, pages 341-350, Roma, Italia, August 2001.

[12] L. K. Poola and J. R. Haritsa. “SphinX: Schema-conscious XML Indexing”. Technical Report RE200104, Indian Institute of Science, Bangalore, India, 2001.

[13] H. V. Jagadish, N. Koudas and D. Srivastava. “On Effective Multi-Dimensional Indexing for Strings”. Proc. of the ACM SIGMOD International Conference on Management of Data, pages 403-414, Dallas, Texas, USA, May 2000.

[14] Q. Li and B. Moon. “Indexing and Querying XML Data for Regular Path Expressions”. Proc. of the VLDB Confernce, pages 361-370, Roma, Italia, August 2001.

[15] Chin-Wan Chung, Jun-Ki Min, Kyuseok Shim. ”APEX: an adaptive path index for XML data”. Proc. of the ACM SIGMOD International Conference on Management of Data, pages 121-132, Madison, Wisconsin, June 2002.

[16] F. Rizzolo and A. Mendelzon. “Indexing XML Data With ToXin”. Proc. of the Int. Workshop on the Web and Databases, pages 49-54, Santa Barbara, California, USA, May 2001.

[17] Alberto O. Mendelzon. “ToX: The Toronto XML Server”. Proc. Int. Database Engineering and Applications Symposium (IDEAS). IEEE CS Press. Edmonton, Canada, July 2002.

[18] Rudolf Bayer and Edward M. McCreight. “Organization and maintenance of large ordered indexes”. Acta Informatica, 1(3):173-189, February 1972.

[19] Beng C. Ooi and K.-L. Tan. “B-trees: Bearing Fruits of All Kinds". IEEE CS Press. Australasian Database Conference 2002, Melbourne, Victoria, January/February 2002.

[20] James Clark and Steve DeRose. “XML Path Language (XPath), version 1.0”, W3C recommendation. Technical Report REC-xpath-19991116, World Wide Web Consortium, November 1999

[21] Andrea Kofler-Vogt and Harald Kosch: “BeTrIS: Efficient Access to MPEG-7 Encoded Data”. Technichal Report TR/ITEC/04/2.10, University Klagenfurt, April 2004