EFFICIENT DATABASE FACILITIES FOR CONTENT-BASED FLASH RETRIEVAL

Bo Feng1 Qing Li1 Jun Yang1 Liu Wenyin2 Jian Zhai1

1Dept. of Computer Engineering and Information Technology
2Dept. of Computer Science
City University of Hong Kong, HKSAR, China
{itfeng, itqli, csliuwy}@cityu.edu.hk yangjun@acm.org jian.zhai@student.cityu.edu.hk

ABSTRACT

Flash™ is experiencing a breathtaking growth and has become one of the prevailing media formats on the Web. Unfortunately, little research effort has been dedicated to the retrieval of Flash movies by content, which is critical to the utilization of the enormous Flash resource. Unlike text or image, a Flash movie contains a number of components in different types such as objects, events, and interactions, with the number ranging from tens to hundreds. How to index and store these components effectively and efficiently in a database is a big challenge. In this paper, we propose a novel star schema for database design and associated facilities, based on the movie’s characteristics at object, event and interaction levels. Experiment results are given to show that our approach can retrieve more relevant movies, efficiently use the storage space and speed up the retrieval process.

Keywords: Flash retrieval, database design, star schema

1. INTRODUCTION

Flash™ is a new type of media format that integrates audio, video, hypertext, graphic, animation and user interaction. Created using Macromedia Flash tools [1], Flash movies are usually embedded in web pages and delivered over the Web. Since its birth in 1997, Flash has experienced an explosive growth throughout the world and has become one of the most popular media formats on the Web. Till October 2002, about 98% web browsers can display Flash movies using Flash Player, according to the latest statistics [2].

A close examination of Flash reveals its intrinsic complexity on three major aspects: (1) a typical Flash movie usually contains heterogeneous media components, including texts, graphics, images, QuickTime videos, sounds and even recursively embedded Flash movies; (2) it has dynamic effects constituted by the spatio-temporal features of its components; and (3) it supports interactivity by allowing users to interfere with the presentation of the movie. This complexity can be summarized into three abstract levels: (1) Object level, which describes the heterogeneous components embedded in a movie; (2) Event level, which depicts the movie’s dynamic effects constituted by the spatio-temporal features of objects; and (3) Interaction level, which models the relationships between user behaviors and the consequential events. These unique characteristics entail a number of research issues regarding content-based Flash retrieval, specifically, the indexing, retrieval methods, and the database accommodation schema for various media components (embedded in movies), their special features, and the forms of user interactions supported. While certain issues (e.g., retrieval of media components) have been thoroughly or partially addressed in previous works, other issues (e.g., database schema design or interactivity) are almost unexplored.

When we search in Google [4], we can find about 800 million Flash movies in the result list. Such a gigantic amount of files make content-based Flash retrieval very hard to be realized using traditional indexing methods. Therefore, an efficient indexing model should be well designed, and the data structure for storing components, effects and interactions of these movies should be very elegant and compact. This issue has been widely discussed in many research works on image processing or video processing, and lots of effective schemes have been proposed. For example, Jane You et al.[5] proposed an agent-oriented approach for effective and efficient visual information retrieval. However, little research work has been conducted on Flash processing. To the best of our knowledge, only Zhai et al.[10] systematically presented a multi-level indexing model for rich media (including Flash) retrieval based on the content. No one has actually investigated the problem of how to design the database structure elegantly and compactly for Flash movies. In this paper, as the first attempt in this direction, we propose a novel database schema design for storing Flash movies, as well as a series of indexing and querying methods.

The rest of the paper is organized as follows. In section 2, we describe the database schema design and algorithms. A multi-module retrieval method for searching Flash movies is given in section 3. A prototypical system for Web-based Flash retrieval is described in section 4, and experiment results are also given in this section to show that our schema design with tailored algorithms is more effective and efficient than other designs. Conclusions and future works are discussed in section 5.

2. SCHEMA DESIGN AND QUERY METHODS

In this section, we describe a novel database schema design for Flash movie management, as well as the algorithms for Flash classification and schema optimization. Firstly, a Flash-to-XML converter is introduced before we discuss the detail of the schema design.

2.1 Flash-to-XML Converter

As a content-based retrieval system, we need to extract all levels of features from a Flash movie and store these features into the database. We call this module a Flash-to-XML Converter. Figure 1 shows its function in our design. It can convert the binary contents of a Flash movie into a series of XML [7] tags, and therefore the different types of components can be easily recognized and stored into the database. There are already some tools which support this conversion, such JavaSWF, a free tool that can be downloaded from [8]. There are two advantages of converting Flash movies into XML formats: (1) XML files are readable and thus convenient for us to understand the internal structure of Flash movies; (2) being a global standard, XML format supports interoperability with other applications.

Figure 1: A Flash-to-XML converter
2.2 Schema Design

2.2.1 Star schema

Based on the motivations presented in section 1, a novel star schema for the Flash database is designed and employed, according to the three levels of Flash movies – object level, event level and interaction level - as shown in Figure 2.

In this schema design, there are totally four tables in the database, one fact table and three dimensional tables. The fact table contains common properties about a Flash movie such as file name, location, components and other essential information. The dimensional tables are object table, event table and interaction table, each of which contains the components of respective types extracted from the Flash movies. The fact table and three dimensional tables are linked to each other through the elements in ObjectList, EventList and InteractList fields. Rather than containing just one ID number in attributes ObjectList, EventList and InteractList in the fact table, a value range (e.g. 1-20) is used to refer to the elements in dimensional tables, indicating that all elements with the IDs within the range are belonging to the same Flash movie. For example, assume there are two Flash movies in the database with the file name “cs.swf” and “shoot.swf” respectively. After converting Flash movies into XML format by the Flash-to-XML converter, two XML files are created which contain understandable tags representing the content of the movies. After that, the Flash parser (see Figure 1) extracts from the XML files their features at object level, event level and interaction level, and the extracted features are then saved into corresponding tables in the database. The fact tuples for these two movies in the Fact Table are shown in Figure 3:

<table>
<thead>
<tr>
<th>FileID</th>
<th>CategoryID</th>
<th>FileSize</th>
<th>Author</th>
<th>UpdateTime</th>
<th>ObjectList</th>
<th>EventList</th>
<th>InteractList</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>cs</td>
<td></td>
<td></td>
<td></td>
<td>1-17</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>shoot</td>
<td></td>
<td></td>
<td></td>
<td>18-44</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The types and features in the three dimensional tables are given in Table 1. Three dimensional tables are referenced to each other through RefObjectID and RefEventID attributes. As an illustration, consider a Flash movie containing a text “sun rising” and a red circle graphic, which, when clicked by a user, will ascend from button to up of the movie like sun rising. The corresponding dimensional tuples for the example Flash movie are shown below:

<table>
<thead>
<tr>
<th>Type</th>
<th>Feature</th>
</tr>
</thead>
<tbody>
<tr>
<td>Text</td>
<td>‘sun rising’</td>
</tr>
<tr>
<td>Graphic</td>
<td>‘red circle’</td>
</tr>
<tr>
<td>Motion</td>
<td>‘ascending’</td>
</tr>
<tr>
<td>Mouse</td>
<td>‘click’</td>
</tr>
</tbody>
</table>

2.2.2 Classified storage

In order to access the Flash movies efficiently, a classified storage is used here. As shown in Figure 2, an attribute “CategoryID” is used to indicate which category the current Flash movie belongs to. We divide Flash movies into five separated categories according to the content of the movie; Game, Advertisement, ECard, MTV and Cartoon, with CategoryIDs ranging from 1 to 5. The classification is done automatically when a Flash movie is crawled from the Web. An overview of the classification algorithm is given in Figure 4, where \( W_x \) is the similarity-weight for each \( x \) indicates.

Table 1: Features for objects, events and interactions.

<table>
<thead>
<tr>
<th>Type</th>
<th>Feature</th>
</tr>
</thead>
<tbody>
<tr>
<td>Object</td>
<td></td>
</tr>
<tr>
<td>ObjectID</td>
<td>ObjectType</td>
</tr>
<tr>
<td>1</td>
<td>Text</td>
</tr>
<tr>
<td>2</td>
<td>Graphic</td>
</tr>
<tr>
<td>Event</td>
<td></td>
</tr>
<tr>
<td>EventID</td>
<td>EventType</td>
</tr>
<tr>
<td>1</td>
<td>Motion</td>
</tr>
<tr>
<td>Interaction</td>
<td></td>
</tr>
<tr>
<td>InteractID</td>
<td>InteractType</td>
</tr>
<tr>
<td>1</td>
<td>Mouse</td>
</tr>
</tbody>
</table>

Figure 4: procedure for category classification
With the attribute CategoryID successfully built up, our Flash movie database is now horizontally partitioned into five parts: Game, Advertisement, ECard, MTV and Cartoon. When a user wants to search for Flash movies, he/she may be asked to select a category under which the retrieval is carried out. This strategy will help accelerate the retrieval to a large extent, because rather than a linear scan of all tuples, only the tuples with selected CategoryID will be scanned. For example, if a user wants to find all Flash movies that contain song “lovestory.mp3” under MTV category, only those attributes whose CategoryID equals to 4 are to be checked to see whether there is an object with type “Sound” and feature “lovestory.mp3”. All other tuples in the database can be skipped.

2.2.3 Optimization with inverted file structure

Although a classified storage has already been applied in the schema design, the number of tuples in each category is still gigantic. In fact, some features such as Rotate or Mouse Action rarely occur in user queries, because few users will form their query like “Find all Flash movies containing a Red Circle Rotating 90° clockwise”. On the other hand, to our knowledge, lots of queries are formed to find Flash movies containing the predefined keywords such as “Happy Christmas” or “Happy Birthday”, or by the file names. Thus with an inverted file structure storing text keywords, we may accelerate the processing of searching such keyword-based queries.

As we know, it is time-consuming to find the matched keyword by string comparison. If the keyword are converted to integers and then compared with a range, the query speed will be improved dramatically. Hence, a lookup table is built up for this purpose, which is called lexicon. The inverted file structure of this lexicon contains two sample tuples is shown below:

<table>
<thead>
<tr>
<th>ID</th>
<th>Keyword</th>
<th>FileID</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>“Happy Birthday”</td>
<td>1,23,48,121,321,516</td>
</tr>
<tr>
<td>2</td>
<td>“The Titanic”</td>
<td>2,89,159,214,450</td>
</tr>
</tbody>
</table>

The FileID here is same as the one in the Fact Table in Figure 2. The comma is used to separate different File IDs. The algorithm for building such lexicon is shown as below:

**Procedure:**

1. Get next tuple T from the Object Dimensional Table
2. If T.ObjectType = “Text”, go to step 1
3. Else if T.Features.Keyword already exists in the lexicon, go to step 3
4. Else add T.Features.Keyword into the lexicon
5. In the Fact Table, find the Flash movie F, where T.ObjectID = F.ObjectID
6. Add F.FileID into the lexicon
7. Add a “;” as the separator after F.FileID
8. Go back to step 1 till no more tuples in the table

**Figure 5:** procedure for building the lexicon

In the same way, such lexicon can be also built for other object types or types in other dimensional tables. Here we only use text keyword lexicon as an example to verify its feasibility and efficiency.

3. MULTI-LEVEL QUERYING MODULES

As a Flash movie is an integration of heterogeneous components, dynamic effects and user interactions, we can retrieve Flash movies at the different levels of its content. A multi-level query engine is designed to decompose user queries into sub-queries at object, event and interaction level and then to integrate and translate the results of the sub-queries into a list of relevant movies. This retrieval method is embodied by the four modules summarized below; detailed elaboration of this method is beyond the scope of this paper, but can be found at [6].

1. **Object retrieval module.** This module accepts the type and features of object as input, and returns a list of objects of the specified type ranked by their similarity to the given features.

   - object-list: SearchObject (o-type, o-feature)

2. **Event retrieval module.** To search events, we need to specify search conditions for not only actions but also objects as the “roles” of the actions.

   - Event-list: SearchEvent (a-type, a-feature, object-list)

3. **Interaction retrieval module.** The retrieval of interactions is conducted by the following function:

   - interaction-list: SearchInteraction (i-type, i-feature, event-list)

4. **Multi-level query engine.** The results returned by individual retrieval modules are objects, events and interactions, while the real target of user queries is Flash movie. The function of the multi-level query engine is to translate the retrieved objects, events and interactions into a list of relevant movies, as defined by the following function:

   - movie-list: Rank (object-list/interaction-list)

To accommodate a user query with multiple search conditions, we need to merge multiple lists of movies retrieved based on each search condition into a single list (of movies) giving the final ranking of similar movies.

- movie-list: Merge (movie-list) where \( \sum \{weight\} \) contains the weight indicating the relative importance of each condition, which is preferably specified by users. If not specified, all the weights are assumed to be 1.

4. EXPERIMENTS AND EVALUATIONS

In this section, we describe an experimental prototype called FLAME we have developed to implement the proposed schema design and the cooperated retrieval methods. An analytical study is conducted to evaluate retrieval performance.

4.1 Application Design

As mentioned, our ultimate goal of this research is to develop a powerful online Flash search engine which supports retrieval of Flash movies based on their semantic features. The features may include keyword description, category, pace and interactivity, or their primitive features of media ingredients, dynamic effects, or user interactions, or a combination of them. A good user interface should allow users to compose various types of queries conveniently and efficiently, as well as to display the retrieved Flash movies with an appropriate layout. Starting from this point, the interface of our experimental system can be displayed in standard web browsers and accessed remotely over the Internet.

**Figure 6** shows the flash retrieval interface, which allows users to compose sophisticated queries by specifying both the semantic features and the primitive features of the objects, events and interactions in the desired movies.
4.2 Response Time Evaluation

In this sub-section, we present an analytical cost model and two tests for processing a query with and without the classification algorithm being applied to the database schema design. As the classification procedure is conducted as a pre-processing step before the components of a Flash movie are stored into the database, the cost of the classification algorithm is not considered here on purpose.

The total cost of processing a query is given by:

\[
\text{Total Cost} = \text{IO Cost} + \text{CPU Cost}
\]

Where \(\text{IO Cost}\) is the cost of performing disk I/Os and \(\text{CPU Cost}\) is the cost for performing computation during query processing. As in [11], we concentrate on the \(\text{IO Cost}\) and discard the \(\text{CPU Cost}\). This is because for very large database applications with huge amount of data access, the \(\text{CPU Cost}\)'s contribution to the \(\text{Total Cost}\) will not be significant. On the other hand, as our database is tuple-based, we can use the total number of tuples been scanned as the \(\text{IO Cost}\).

Table 2 shows the total numbers for different elements in our database. For the ease of evaluation, we choose 1000 as the number of Flash movies to be used for tests.

<table>
<thead>
<tr>
<th>Category (ID)</th>
<th>No. of Movies (tuples in Fact Table)</th>
<th>No. of Objects (tuples in Object Table)</th>
<th>No. of Events (tuples in Event Table)</th>
<th>No. of Interactions (tuples in Interaction Table)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Game (1)</td>
<td>217</td>
<td>8556</td>
<td>6944</td>
<td>901</td>
</tr>
<tr>
<td>Adv (2)</td>
<td>189</td>
<td>7614</td>
<td>6197</td>
<td>714</td>
</tr>
<tr>
<td>ECard (3)</td>
<td>160</td>
<td>6761</td>
<td>5579</td>
<td>656</td>
</tr>
<tr>
<td>MTV (4)</td>
<td>251</td>
<td>9975</td>
<td>8954</td>
<td>1072</td>
</tr>
<tr>
<td>Cartoon (5)</td>
<td>183</td>
<td>8116</td>
<td>8397</td>
<td>759</td>
</tr>
<tr>
<td>(Total)</td>
<td>1000</td>
<td>41022</td>
<td>35711</td>
<td>4102</td>
</tr>
</tbody>
</table>

Details for the two evaluations are given below:
(a). Without classification algorithm. In this situation, all tuples in the fact table or dimensiona tables need to be scanned for the respective search condition. Thus the \(\text{Total Cost}\) is the number of tuples in the fact table or the dimensional table.

\[
\text{Total Cost} = \text{IO Cost} = \text{TS Dimensional Table} \quad (1)
\]

(b). With classification algorithm. When the user specifies one or more categories in which the retrieval is conducted, only tuples with the respective CategoryID and the corresponding tuples in the dimensional tables need to be scanned. Here we define the \(\text{IO Cost}\) as follows:

\[
\text{IO Cost} = \text{TS Fact Table} + \text{TS Dimensional Table} \quad (2)
\]

where \(\text{TS Fact Table}\) is the number of tuples scanned for the CategoryID in the fact table, which is the total number of tuples, and \(\text{TS Dimensional Table}\) is the number of tuples scanned for the search condition in the dimensional table. Furthermore,

\[
\text{Total Cost}_2 = \sum \text{IO Cost}_i \times C_i, \quad (C_i = 0 \text{ or } 1) \quad (3)
\]

where \(C_i\) is a binary flag for the five categories, 1 standing for choosing that category and 0 standing for not choosing that category. At last, we can calculate the savings as:

\[
\text{Total Saving} = (\text{Total Cost}_1 - \text{Total Cost}_2) / \text{Total Cost}_1 \quad (4)
\]

Let us consider the following query as an example:

\[
\text{Rank (SearchObject (text, "Great Wall"))}
\]

In situation (a), \(\text{Total Cost}_1 = 41022\), which is the total number of tuples in Object table. In situation (b), assume the query gets parsed and resolved\(^\text{4}\) by using ECard (3) as the searching category, which means only \(C_3 = 1\), then \(\text{Total Cost}_2 = 1000 + 6761 + 16877\), and \(\text{Total Saving} = (41022 - 1000 - 6761 - 16877) / 41022 = 0.8101 \approx 81.08\%\).

For the same example but suppose the user query gets resolved by using ECard (3) and Cartoon (5) as the searching categories, i.e. \(C_3\) and \(C_5\) are set to 1. Then \(\text{Total Cost}_2 = 1000 + 6761 + 1000 + 8116 + 16877\), and \(\text{Total Saving} = (41022 - 16877) / 41022 = 0.5885 \approx 58.85\%\).

From the results above, we can see that the classification fragmentation schema can accelerate the retrieval speed effectively. Moreover, the more specific category used for the query, the faster the retrieving speed is.

5. CONCLUSION AND FUTURE WORK

In this paper, we have presented a novel star schema design for Flash database based on object level, event level and interaction level, as well as a classified storage and a lexicon algorithm to speed up the retrieval process. An experimental prototype for Web-based Flash retrieval has been implemented to verify the feasibility and effectiveness of the whole design, upon which a performance evaluation with and without classification has been conducted, with the results demonstrating a big difference in terms of speed.

In our future work, more features and relationships among different dimensional tables will be incorporated into the database schema design. To reach a better performance, the Flash classification algorithm and the multi-level query method can also be improved in conjunction with a multi-level indexing mechanism [10]. Last but not least, the research on Flash database can be generalized to the retrieval of other types of multimedia representations largely existing on the Web, such as PowerPoint, SMIL, etc.

Acknowledgement: The work described in this paper was substantially supported by a grant from CityU (Project no.7001384).

REFERENCES


[7]. Extensible Markup Language (XML). http://www.w3.org/XML/


[9]. Synchronized Multimedia Integration Language (SMIL). http://www.w3.org/AudioVideo


\(^4\)The knowledge on mapping the query to the most relevant categories is maintained by the database, and is utilized transparently from the user’s angle. Details on this aspect are omitted in this paper due to space limit.