An Interactive Query Implementation over High Precision Progressive Query Scheme

Serkan Kiranyaz and Moncef Gabbouj

Institute of Signal Processing, Tampere University of Technology, Tampere, Finland

Abstract The recently proposed Progressive Query method is a dynamic retrieval technique, which is mainly designed to bring an effective solution especially for queries on large-scale multimedia databases and furthermore to provide periodic fractional query retrievals along with the ongoing query process. In this way it achieves a series of sub-query results that will eventually be converging to the full-scale search retrieval in a faster and with no minimum system requirements. Due to its pre-emptive design over a single process, the precision of its time periodicity suffers from the interference of other CPU consuming, heavy processes running in the same system. In this paper we propose a High-Precision Progressive Query scheme based on a parallel processing architecture. Such an approach does not only improve the periodic accuracy of the progressive sub-queries, it furthermore enables an implementation of a novel query scheme providing full interaction with user and immediate retrieval response. This Interactive Query scheme hence becomes one of the major modes of the Progressive Query method under MUVIS framework. Experiments with High Precision Progressive Query show an improved timing accuracy and furthermore with the proposed Interactive Query mode the user is now able to see any intermediate retrieval result instantly whenever desired.

1 Introduction

The usual approach for multimedia indexing is to map database items such as images, video and audio clips into some high dimensional vector space, the so called feature domain [1]. The feature domain may consist of several types of features such as visual, aural, motion, etc. as long as the database contains multimedia items from which those particular features can be extracted. Among so many variations, careful selection of the feature sets allows capturing the semantics of the database items. Especially for large multimedia databases the number of features extracted from the raw data is often kept large due to the naïve expectation that it helps to capture the semantics better. Content-based similarity between two database items can then be assumed to correspond to the (dis-) similarity distance of their feature vectors. Henceforth, the retrieval of a similar database items with respect to a given query (item) can be transformed into the problem of finding such database items that gives feature vectors, which are close to the query feature vector. This is so-called query-by-example (QBE), which is one of the most common retrieval schemes. The exhaustive search based QBE operation is so called *Normal Query* (*NQ*), and works as follows: using the available aural or visual features (or both) of the queried multimedia item (i.e. an image, a video clip, an audio clip, etc.) and all the database items, the similarity distances are calculated and then merged to obtain a unique similarity distance per database item. Ranking the items according to their similarity distances (to the queried item) over the entire database yields the query result. *NQ* is costly and CPU intensive especially for large-scale multimedia databases since the number of similarity distance calculations is proportional with the database size. Furthermore any abrupt stopping during the query process will cause total loss of retrieval information and essentially nothing can be saved out of query operation so far performed.

In order to eliminate such drawbacks and provide a faster query scheme, *t*he *Progressive Query* (*PQ*) [4] has been recently proposed. It basically forms periodic *Progressive Sub-Query* (*PSQ*) retrievals to the user and allows user to interact with the ongoing query process. It has a pre-emptive linear estimator for adaptive setting of the periodic sub-set sizes on a single process and therefore, the accuracy of the estimated time periodicity suffers from other processes and might yield up to 20% of timing shifts from the required period value, t_p, which can be avoided by changing the design into a parallel processing architecture. In this paper we first propose a *High-Precision Progressive Query* (*HP-PQ*) scheme based on two parallel processes and a high resolution timer. Such an approach does not only improve the precision of the time periods of the progressive sub-queries, it further forms the necessary basis for the implementation of a novel query scheme, the so called *Interactive Query* (*IQ*), providing a full interaction with the user. Via using *IQ*, the user can see the next (progressive) retrieval results whenever required and without any noticeable delay. In addition to the periodic approach (*HP-PQ*) the so called *Interactive Query* (*IQ*) hence becomes one of the major modes of the *Progressive Query* retrieval scheme under MUVIS [2], [3] framework.

The rest of this paper is organized as follows: in section 2 we introduce a brief overview about the generic philosophy behind *PQ* and its innovative properties with respect to other common query methodologies. The new *PQ* method, *HP-PQ* and the proposed *IQ* design over it are covered in section 3. Section 4 presents the experimental results and some sample demonstrations. Finally conclusions are drawn in section 5.

2 *Progressive Query* – An Overview

The contemporary *PQ* scheme is designed over periodic sub-queries as shown in the top part of Fig. 1 with a user defined period value ($t = t_p$). Among other traditional query techniques such as exhaustive search based *NQ* that can be only used for databases without any indexing structure or *kNN* and range queries for indexed databases, *PQ* presents significant innovative properties as follows:

- It is an efficient technique, which works within both indexed and non-indexed databases onto which it is the unique query method that may provide "faster" retrievals (than *NQ*).
- It is also the unique query method that provides "Browsing" capability between instances (PSQ) of the ongoing query. Hence the user can browse among the PSQ retrievals in any time, i.e. during or after the query process.
- In databases without any indexing structure, it achieves several improvements such as loose system requirement (in terms of memory, CPU power, etc.), "early and even better" retrieval results, user-friendly query accessibility options (i.e. query can be stopped in case satisfactory results obtained, PSQ retrievals can be browsed any time, etc.), reduced "overall" timing (in case PQ is totally completed), etc.
- The most important advantage above all is that it provides continuous user interaction with the ongoing query operation. The user can see the results so far obtained, can immediately evaluates them and performs "relevance feedback" into the system or simply wastes no time if satisfactory results are obtained so far (query stop).
- It can also be applied to indexed databases efficiently (to get the most relevant retrieval results in the fastest possible way) and in this case it shows a "dynamic *kNN* - range query" like behavior where *k* (or ε) increases gradually with time and hence the user can have the advantage of assigning it by seeing (and judging) the results. This is obviously a significant advantage with respect to a traditional *kNN* or range queries since the user usually does not know a "good" *k* value (or ε value) prior to a query operation.

The details such as atomic and periodic sub-query formation, fractional sub-queries and fusion operation, etc. about the single thread (process) and pre-emptive design of *PQ* can be obtained from [4].

3 New *PQ* Modes*: HP- PQ* versus *IQ*

The proposed *PQ* modes share a similar architecture. So we will first present *HP-PQ* implementation details in the next sub-section and then *IQ* will be presented accordingly.

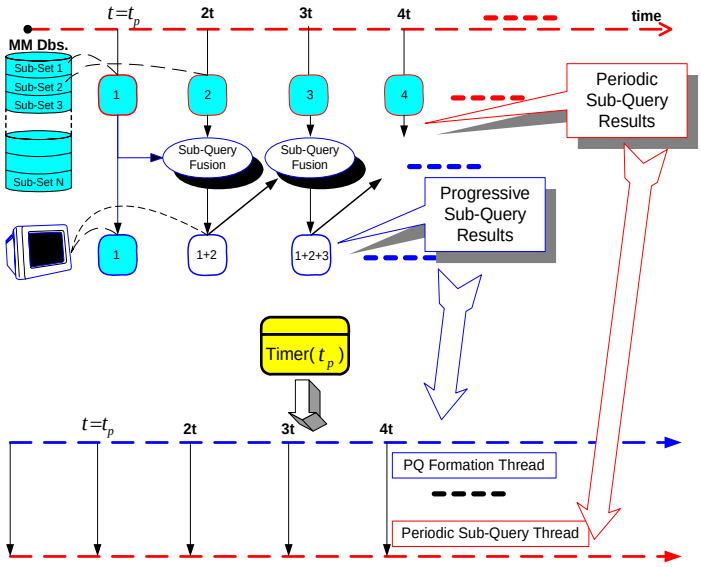

Fig. 1 *HP-PQ* Overview.

3.1. HP- PQ – A Multi-threaded Approach

Significantly higher precision on the *PQ* period can basically be achieved by reforming the traditional design of (single threaded) *PQ* into a multi-thread (parallel processing) basis. The entire *PQ* process is divided between two threads controlled with a timer to perform the following tasks in parallel:

- **Periodic Sub-Query Thread:**
  - Load features from disc into memory.
  - Calculate similarity distance of each item with respect to query item.
  - Sort items within periodic sub-query set.
- ***PQ* Formation Thread:**
  - Suspend Periodic Sub-Query thread when timer kicks.
  - Apply sub-query fusion to form next *PSQ*.
  - Release Periodic Sub-Query thread.
  - Render next *PSQ* retrievals on screen.

Fig. 1 illustrates the how the multithread approach is adapted over the *PQ* implementation in run-time to perform *HP-PQ*. So the main idea is to leave all the CPU consuming tasks to the periodic sub-query thread and keep the other (*PQ* Formation) thread in suspended mode until the timer kicks. Once the timer activates the *PQ* Formation thread then it can

immediately form the *PSQ* retrieval only with a single fusion operation and renders the retrieval results on the screen. By this way the timing accuracy of the *PSQ* retrievals can be within a close proximity (in one or few milliseconds) of the timer resolution. After the first few periodic *PSQ*, the timer can further be adapted according to offset (extra) time to improve the timing accuracy.

3.2. Interactive Query over HP-PQ

HP-PQ runs on two parallel processes: *PQ* formation and periodic sub-query threads. The former is a lightweight process, which is only activated by a timer to fuse the last periodic sub-query to the previous *PSQ* retrieval and show the new *PSQ* results to the user. Once it completes its run-time, it resumes to the idle status and let the latter thread proceeds with the rest of the database items. So in *HP-PQ* the timer has the entire control to determine when the next *PSQ* results to be shown to the user. Such a design can thus be easily adapted to establish an *IQ* operation by excluding the timer and instead giving the full control to the user, that is, the control to get (render) the retrieval results whenever required and without any noticeable delay (immediate response). In the design shown in Fig. 1, if the timer is replaced by the user command initiated from the GUI of the application where *PQ* is active, *IQ* can thus be achieved.

The new *PQ* modes proposed in this paper are developed under MUVIS [2], [3] framework and so far used as the primary retrieval scheme within *MBrowser* application. Fig. 2 shows a snapshot of the *MBrowser* GUI where an *IQ* operation is active for a query image (shown on the top-left side) in Corel image database with 60K images and the user already interacted with the process to retrieve the first two *PSQs* via *IQ* knob. The basic query parameters such as *PQ* mode (*IQ* or *HP-PQ*), query genre (aural or visual), *PQ* update period (only for *HP-PQ*), the sub-feature selection among database (visual and aural) features and their individual parameters (i.e. feature weights), etc. can be set prior to a query operation. When *IQ* is used, the user can get the next *PSQ* retrievals anytime by simply pressing the *IQ* Knob or choosing "Update Query" option on the query menu. The knob can also be used to browse among the *PSQ* results already retrieved whilst the query process is still carried out in parallel (via periodic sub-query thread) without interfering the browsing operation performed by the user. When the user updates the ongoing *PSQ*, the retrieval results are then presented page by page. Each page renders 12 ranked results in the descending order (from left to right and from top to bottom) and the user can browse the pages back and forth using the *Next* and *Prev* buttons on the bottom-right side of the figure (the first page with 12-best retrieval results is shown on the right side of Fig. 2). When the user finds the *PSQ* results satisfactory, then he/she can stop *IQ* from the query menu (i.e. "Stop Query").

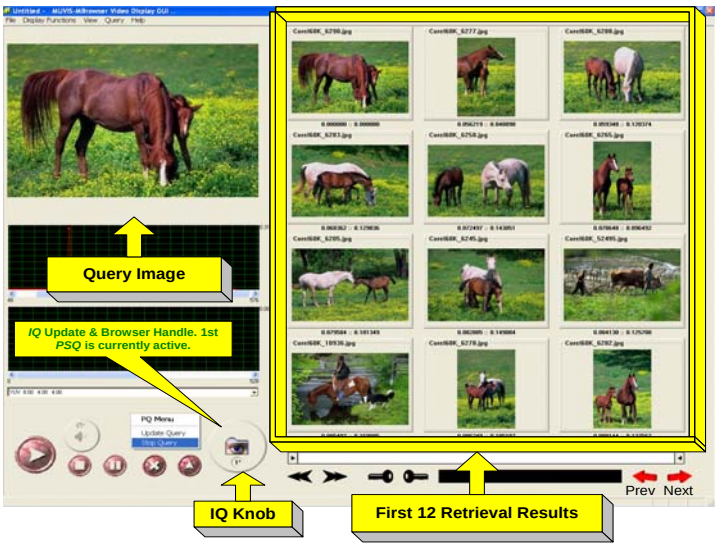

Fig. 2 The GUI of MUVIS-*MBrowser* application with an with an ongoing *IQ* process.

4 Experimental Results

In the experiments performed in this section, we used Corel database with 60K images. All experiments are carried out on a P5 1.8 GHz computer with 1024 MB memory. In order to have unbiased experimental evaluations, each query experiment are performed using the same queried multimedia item with the same instance of *MBrowser* application. The evaluations of the retrieval results by *PQ* are performed subjectively using ground-truth method

Most of the results, such as memory, speed and overall retrieval time, etc. presented for the single-thread *PQ* implementation in [4] are also valid for the new *PQ* modes presented in this paper. The outcome of the new *PQ* implementation scheme, *HP-PQ*, differs only in the timing accuracy of the *PSQ* retrieval times. We present an experiment verifying the timing precision of *HP-PQ* as compared to single-thread *PQ* as shown in Fig. 3.

For the purpose of demonstrating the user interaction options along with the *PSQ* retrieval time measurements for *IQ*, Fig. 4 presents an *IQ* operation in Corel database. The user makes two updates within 3 seconds and then stops the *IQ* operation since the retrieval results are so far satisfactory and hence there is no reason to carry on the operation. As shown in the figure, in case the user would not have stopped the *IQ* operation, the entire operation (full scan) would take more than 18 seconds and the final results would be identical to the second update of *IQ*. Furthermore note that some of the relevant results in the first update disappeared in the second one, so the user can have the opportunity to browse among the updates during or after the ongoing *IQ* operation to visit all the relevant "items of interest".

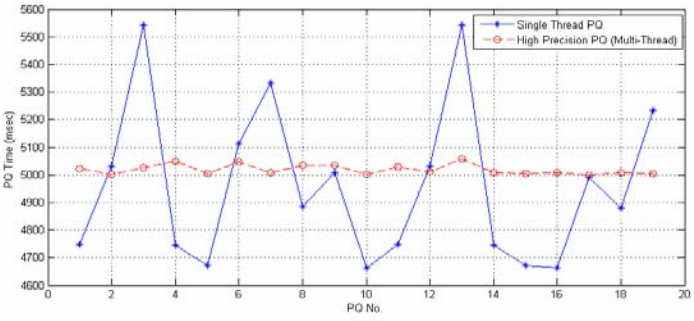

Fig. 3 *PSQ* Retrieval times for single and multi threaded (*HP*) *PQ* schemes with $t_p = 5\,\text{sec}$.

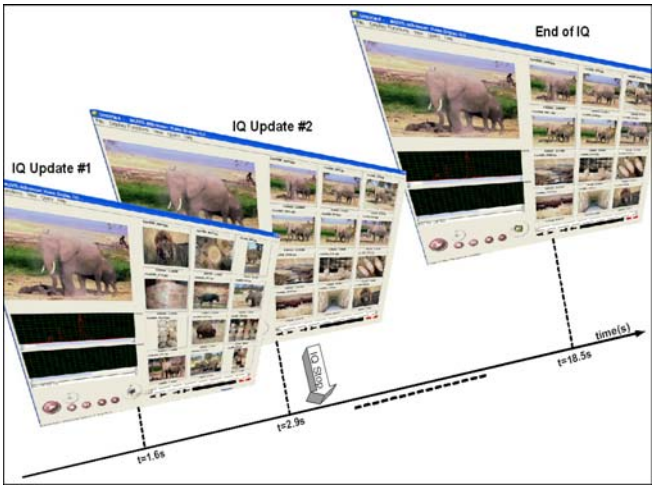

Fig. 4 *IQ* operation in Corel database.

Finally when there is an efficient indexing structure available in the database, both *IQ* and *HP-PQ* can be used to retrieve the relevant items in the earliest possible time. For this purpose we index the Corel database using *Hierarchical Cellular Tree* (*HCT*) indexing technique [5] and perform 100 *HP-PQ* operations with and without (i.e. sequential scan) using *HCT* indexing body. We used $t_p = 1\,\text{sec}$ and thus measured the query time to retrieve the relevant images (either all or only one miss allowed) among the first (highest ranked) 12 results. The query time histograms are drawn according to the measurements and shown in Fig. 5. As expected *PQ* over *HCT* achieves the earliest retrieval times where almost half of the retrievals are achieved within one second and only 7 (out of 100) *PQ* over *HCT* experiments yield retrieval times exceeding 4 seconds. As a traditional query mechanism, *NQ* in general provides the slowest retrieval speed, almost all in 18 seconds, only after the full-scan search is completed over the entire database. Sequential *HP-PQ*, on the other hand provides a significantly varying scheme since it is mainly used for the databases with no indexing structure (hence indexing information is not used at all) and the majority of the query results provides the required amount of relevant items after 11 or more seconds.

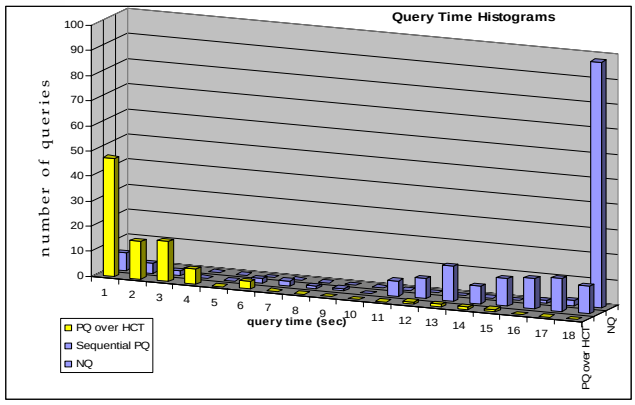

Fig. 5: The query time histograms obtained from 100 *HP-PQs* (using both modes) and *NQ*s in Corel database.

5 Conclusions

PQ is a query method recently developed to provide instantaneous and faster retrievals along with the ongoing query process. It promises several advantages for multimedia databases with or without an indexing structure; however due to its traditional design based on single thread execution, it suffers from the variations in timing accuracy. In this paper we proposed a new *PQ* scheme, *HP-PQ*, which is based on a parallel processing architecture and provides a high precision on the *PSQ* periods. Furthermore it allows a novel *PQ* mode, *IQ*, which maximizes the user interaction with the ongoing query process in such a way that the user can see the retrieval results instantly whenever he/she wants. During the time the user evaluates and browses the available *PSQ* retrievals, *IQ* scanning process proceeds in parallel more and more through the database and therefore, any time lost due to the interference in between is prevented.

References

[1] S. Deb, Y. Zhang, "An Overview of Content-based Image Retrieval Techniques", 18th International Conference on Advanced Information Networking and Applications (AINA'04) Volume 1, pp. 59, 2004.

[2] MUVIS. http://muvis.cs.tut.fi/

[3] S. Kiranyaz, "Advanced Techniques for Content-based Management of Multimedia Databases", PhD Thesis, Pub. 541, Tampere University of Technology, Tampere, Finland, June 2005.

[4] S. Kiranyaz, M. Gabbouj, "A Novel Multimedia Retrieval Technique: Progressive Query (Why Wait?)", *IEE Proc. Vision, Image and Signal Processing*, vol. 152, pp. 356-366, May 2005.

[5] S. Kiranyaz, M. Gabbouj, "A Dynamic Content-based Indexing Method for Multimedia Databases: Hierarchical Cellular Tree", *In Proc. of IEEE International Conference on Image Processing, ICIP 2005*, Paper ID: 2896, Genova, Italy, September, 2005.