Fast Audio Fingerprinting System Using GPU and a Clustering-Based Technique

Chahid Ouali, *Member, IEEE*, Pierre Dumouchel, *Member, IEEE*, and Vishwa Gupta, *Life Senior Member, IEEE*

***Abstract*—In this paper, we present our audio fingerprinting system that detects a transformed copy of an audio from a large collection of audios in a database. The audio fingerprints in this system encode the positions of salient regions of binary images derived from a spectrogram matrix. The similarity between two fingerprints is defined as the intersection of their elements (i.e. positions of the salient regions). The search algorithm labels each reference fingerprint in the database with the closest query frame and then counts the number of matching frames when the query is overlaid over the reference. The best match is based on this count. The salient regions fingerprints together with this nearest-neighbor search give excellent copy detection results. However, for a large database, this search is time consuming. To reduce the search time, we accelerate this similarity search by using a graphics processing unit (GPU). To speed this search even further, we use a two-step search based on a clustering technique and a lookup table that reduces the number of comparisons between the query and the reference fingerprints. We also explore the tradeoff between the speed of search and the copy detection performance. The resulting system achieves excellent results on TRECVID 2009 and 2010 datasets and outperforms several state-of-the-art audio copy detection systems in detection performance, localization accuracy and run time. For a fast detection scenario with detection speed comparable to the Ellis' Shazam-based system, our system achieved the same min NDCR as the NN-based system, and significantly better detection accuracy than Ellis' Shazam-based system.**

***Index Terms*—Audio fingerprint, content-based copy detection, fast search, parallel processing, GPU, clustering.**

I. Introduction

ADVANCEMENTS in information technology since the beginning of the digital revolution in the late 1950 s have incorporated multimedia in all aspects of our life. Specifically, the innovations in telecommunications, signal compression, data transmission and the increase in computing and storage capacities have made the production, distribution and reproduction of multimedia content not only possible, but also a widespread phenomenon ubiquitous in our social and professional lives. The increasing popularity of video sharing web sites, such as YouTube, Dailymotion and Metacafe, show the extent of this reality. For instance, YouTube claims that, in March 2015, 300 hours of videos were uploaded every single minute [1]. This huge data traffic also implies serious copyright issues. In fact, a large number of the uploaded content may be illegal copies of digital material protected by copyright law. To deal with this, YouTube has invested millions of dollars into their Content ID copyright management system, and has paid out over 1 billion to partners who have chosen to monetize their claims using Content ID [1]. File sharing networks (peer-to-peer) have cost the music industry billions in economic losses. According to the Recording Industry Association of America (RIAA), 30 billion songs were illegally downloaded on file-sharing networks between 2004 and 2009 [2]. These facts have made copyright infringement one of the biggest issues that hosting web sites, file sharing networks and similar services have to deal with to avoid copyright violation lawsuits.

Content-Based Copy Detection (CBCD) has been used to avoid copyright infringements. It extracts relevant features (called signatures or fingerprints) from a candidate copy and then compares them against fingerprints of the original content. To avoid detection, audio signals are subjected to various kinds of transformations that make robust fingerprint extraction challenging. The robustness of fingerprints against audio transformations is an important element of any copy detection system. An ideal fingerprinting system should be able to detect a copy of an original audio file regardless of the complexity of the transformations [3], [4]. Another essential requirement is the search efficiency. The speed of the fingerprints search is becoming increasingly important due to the large volume of data. The search of an audio against a large set of fingerprints should be at least real time.

In this paper, we describe a fast and robust audio fingerprinting system suitable for a wide variety of applications including copyright control, advertisement tracking, and music identification. The robustness of this system is achieved by generating fingerprints that encode the positions of salient regions of binary images derived from a spectrogram matrix [5]. These fingerprints have shown their robustness against several audio distortions. However, the similarity search is time consuming due to the large amount of data and the high dimensional representation of the fingerprints. We propose an approach based on the combination of hardware and software techniques that speed up this similarity search by several orders of magnitude. Specifically, we use a Graphics Processing Unit (GPU) to perform the similarity computations over millions of fingerprints in parallel. We extend our work in [6] that introduced efficient GPU implementations of two similarity search algorithms, by

Manuscript received November 25, 2015; revised March 01, 2016; accepted March 08, 2016. Date of publication March 11, 2016; date of current version April 29, 2016. The associate editor coordinating the review of this manuscript and approving it for publication was Dr. Matthew E. P. Davies.

C. Ouali and P. Dumouchel are with the Département de Génie Logiciel et des Technologies de l'Information, École de Technologie Supérieure (ETS), Montreal, QC H3C 1K3, Canada (e-mail: chahid.ouali.1@ens.etsmtl.ca; pierre.dumouchel@etsmtl.ca).

V. Gupta is with the Department of Speech Recognition, Centre de Recherche Informatique de Montréal (CRIM), Montreal, QC H3N 1M3, Canada (e-mail: vishwa.gupta@crim.ca).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TASLP.2016.2541303

proposing here a two-step search algorithm based on a clustering technique that reduces the number of fingerprint comparisons significantly. This two-step search reduces the average run time per query from 7 hours (using the CPU) to a few seconds. We evaluate our approach on TRECVID 2010 dataset, and we validate the results using TRECVID 2009 dataset. In addition, we compare our system to several state-of-the-art fingerprinting systems.

This paper is organized as follows: Section II reviews previous approaches in audio CBCD. Section III gives an overview of our audio fingerprinting system including the feature extraction and the search algorithm. We present the GPU implementation, the clustering-based technique and the two-step search in Sections IV, V and VI, respectively. In section VII, we give experimental details and compare our system to several state-of-the-art CBCD systems. Section VIII recapitulates our contributions.

II. Related Work

A review of audio copy detection systems shows three main approaches to accelerate fingerprint search: binary search, hashing-based search and approximate search. The energy difference fingerprint, where a binary fingerprint encodes the energy differences along the frequency and the time axes, figures among the fastest CBCD systems [7]–[10]. The binary representation of the fingerprints makes the search very fast. However, this fast search results in a modest performance compared to other methods. In [11], regions around selected points from the maxima in the Mel-filtered spectrum are encoded to generate binary fingerprints. Compared to [8], this approach improved significantly the detection accuracy while maintaining a fast search. In [12], local regions of the spectrogram image are transformed into a set of 32 bit vectors, and a classical hash table is used to perform the search. In the Shazam system [13], several time-frequency points are chosen from the spectrogram. Compact signatures representing peak pairs are then generated to form fingerprint hashes that allow very fast search. Approximate searching techniques such as Locality Sensitive Hashing (LSH) are used in several works to accelerate the search. In [14], [15], wavelets with the largest magnitude are selected from spectrogram, and LSH is used to accelerate fingerprint similarity search. Although more robust than [8] and [13], this system is computationally very expensive [11]. A comparative study of [12], [13] and [15] in terms of detection accuracy and computation time can be found in [16]. In [17], the Weighted Audio Spectrum Flatness (WASF) is used as audio features, and LSH is adapted to compute the dissimilarity between two WASF features. LSH is used in many works to accelerate the search, but it is slower than the hashing-based search [12].

Recently, Graphics Processing Units (GPUs) have been used to accelerate scientific computations. Using GPU to accelerate large-scale applications became easier with NVIDIA's CUDA platform. Several GPU implementations of widely used algorithms such as k-nearest neighbor [18] and LSH [19] are available and can be used for audio copy detection. A GPU implementation of the Metric Permutation Table algorithm [20] speeds up the search of digital images. In [21], Mel-Frequency Cepstral Coefficients (MFCCs) plus energy and its delta coefficients are used as audio features. A GPU implementation of the nearest neighbor search between the reference and query fingerprint is then described in [22], where the copy detection algorithm has been modified to perform advertisement detection. Compared to its CPU implementation, this GPU implementation improves speed by a factor of 70. Similar GPU based nearest neighbor search is used in [23], [24] to search through millions of fingerprints. However, these papers do not include a description of the GPU implementations. A parallel implementation of [13] is introduced in [25] and tested over a large database of more than 11,600 hours of audio. A GPU is used to parallelize two parts of the system leading to an overall speedup of a factor of 5. The authors also explored the use of three GPUs instead of only one allowing them to further improve performance by a factor of 3 on some parts of the system. In another work, the computation of the cross-correlation between two audio windows is accelerated using a GPU [26]. The database used to test this algorithm is very small (1 hour), and the GPU lead to a moderate improvement by a factor of 2 (compared to the CPU implementation).

In Another technique that reduces search time and complexity of [8] is proposed in [27]. Starting from the fact that search time is related to the size of the database, the authors partitioned the database of 100,000 songs into 10 sub-databases. The search then executes 10 independent processes on different machines. Similar to [27], [28] divides the fingerprints database into several parts, and the search algorithm is executed in parallel based on the Message Passing Interface (MPI) standard.

Clustering techniques have been used in several works to avoiding exhaustive search. In [29], binary fingerprints of the reference videos are grouped into *k* different clusters, and only fingerprints that belong to the cluster closest to the query fingerprint are searched to find a match. The algorithm continues to examine other clusters if a match is not found. The problem in using this strategy is the possibility of visiting all the *k* clusters before a match is found resulting in an exhaustive search. In another related work, a bag of audio words model is proposed to group MFCCs and RASTA-PLP [30] features into different feature spaces [31]. This model combined with inverted file retrieval makes large-scale audio copy detection possible in real time. However, the results achieved by this system are not motivating, especially when the queries are distorted by adding irrelevant speech. The technique in [31] is inspired by the bag of visual words which is applied in several video copy detection systems [17], [32]–[34].

III. System Overview

In this section we describe our audio fingerprinting system shown in Fig. 1. In a typical CBCD task, we search for a query audio in a database of reference audio files (containing copyrighted or original audio content) to see if the query is a copy of one of these reference audio files. For this task, we first extract robust audio fingerprints [5] from these reference audio files. To extract these fingerprints, we first transform the audio signal into a spectrogram. We convert the resulting spectrogram into

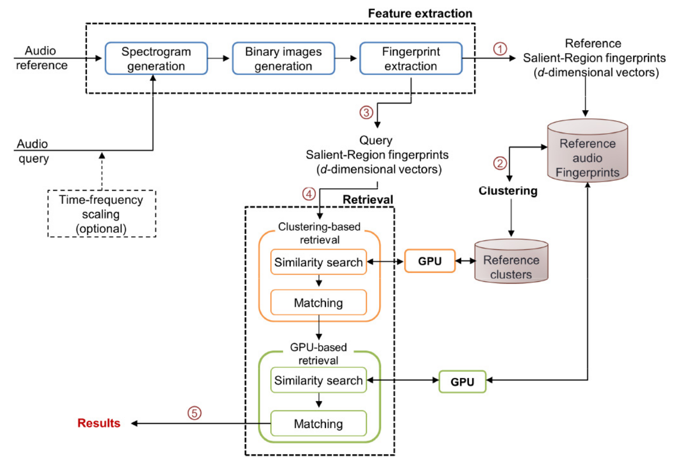

Fig. 1. System overview.

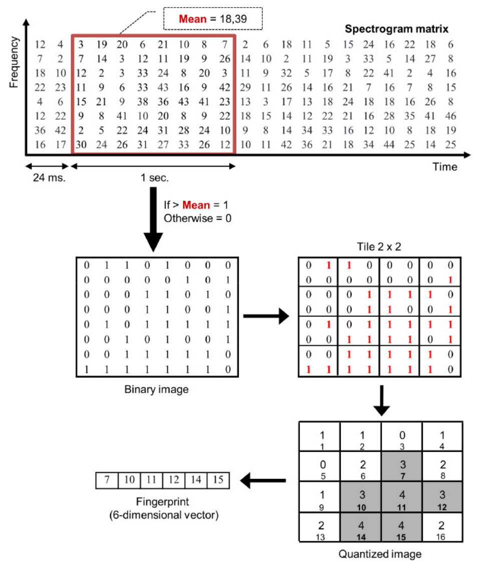

Fig. 2. Feature extraction with 16 tiles (D=16) and 6 salient tiles ($d = 6$) from quanized binary image derived from the spectrogram matrix.

a set of 2-D binary images. Finally, we extract the top-d salient regions from each binary image. Each *Salient-Regions* fingerprint is represented by a d-dimensional vector, which contains the positions of the selected salient regions only, and stored into a reference fingerprint database.

Once all reference fingerprints have been generated, a k-means like clustering algorithm groups the reference fingerprints into different clusters. These clusters are then used to accelerate the fingerprint search.

Query fingerprints are extracted in the same way with an optional time-frequency scaling step (resampling) that reduces the speed difference between the query and the reference.

Fingerprint retrieval is a two-step search: clustering-based retrieval followed by a GPU-based retrieval. This two step retrieval reduces the search time by several orders of magnitude. In this two-step search algorithm, a similarity search is performed using either the reference clusters (in case of clustering-based retrieval) or the original references (in case of GPU-based retrieval) to find the closest query fingerprint for each reference fingerprint. This similarity search associates with each reference fingerprint a query frame number corresponding to the best matching query fingerprint. The similarity search is followed by a matching step that associates a matching score between each reference file and the query. This matching step produces the final results and determines if the query is a copy of a reference audio file.

A. Feature Extraction

The feature extraction step is shown in Fig. 2. The spectrogram matrix in this figure represents the intensity of the signal at any given time and frequency. The spectrogram is obtained using a MATLAB routine in the VOICEBOX toolbox [35]. This routine applies a Hamming window of length 96 ms and computes the short time Fourier transform in this window every 3 ms. The spectrogram is reduced to 257 frequency bins in the frequency range from 500 Hz to 3 kHz. Several audio fingerprinting systems [8], [14], [36] reduce the spectrogram to 33 bins while using smaller frequency range (e.g. 300 Hz to 2 kHz). Unlike these works, we use larger range to obtain more detailed shape of the signal and a more discriminative spectrogram image. Adding more information to the spectrogram here does not increase the complexity of our system, since this spectrogram will be converted into compact signatures. These parameters are not critical to our system, and changing their values to a certain degree will not affect the system performance[1].

Each spectrogram matrix is composed of 333 consecutive spectrograms derived from 96 ms windows with a frame advance of 3 ms. Each spectrogram matrix has $333\,\text{frames} \times 257\,\text{frequency bins}$, one spectrogram matrix computed every 24 ms. A considerable number of audio fingerprinting systems [8], [21], [37] use a window shorter than 1 sec (tens of milliseconds). The 1-sec window size is chosen to ensure that the spectrogram matrix contains enough information to be discriminative. As we increase the window size we obtain more discriminative frames. However, a long window frame makes the detection of small queries difficult. For example, TRECVID datasets include queries as short as 3 seconds in duration. In such a case, a 3-sec window size will generate only one fingerprint, and just one fingerprint may make the detection of such short queries difficult.

Second, the large overlap (1-sec frame length with 24 ms frame advance) is chosen to overcome the lack of synchronization between the query and the reference. In fact, a large frame advance can prevent matching of query and reference frames when the start of the query is not synchronized with the start of the reference. Using a small frame advance has two disadvantages. First, short frame advance may generate identical fingerprints for successive frames. This could be an issue for some audio fingerprinting systems such as [23] and [24]. We showed in [5] that our feature extraction scheme avoids this problem and generates different fingerprints for successive

[1]Our experiment on TRECVID 2010 dataset using 100 different queries showed that our system missed 4 more queries when tested with 33 frequency bins in the range between 300 Hz and 2 kHz compared to 257 frequency bins in the range from 500 Hz to 3 kHz used in this paper.

frames. Second, the number of generated fingerprints increases as we use smaller frame advance, increasing the search time. We think that using long frame advance may negatively impact the performance of our system more than using small frame advance. In fact, the proposed two-step similarity search represents a good solution for reducing the search time, allowing us to generate higher number of fingerprints.

The second step of the feature extraction scheme is to convert each 1-sec window (i.e. the 1-sec spectrogram matrix inside the red rectangle in Fig. 2) to a binary image: we compute the mean intensity value of this 1-sec spectrogram matrix, and then replace the intensity values by either 0 or 1 using the following strategy: if the intensity is greater than the mean then it is replaced by a 1, otherwise it is replaced by a 0. Thus, each 1-sec window is transformed into a 2-D binary image, where the intensities that are greater than the mean are replaced by 1 regardless of their real intensity values, while getting rid of intensity values smaller than the mean. This binary representation makes the fingerprint invariant to relative intensity value changes (e.g. overall amplification or reduction of energy, equalization, etc.).

We also tried the median of the intensity values as a threshold. We tested our system using several queries (we compare the query to the corresponding correct reference to count the number of matching frames). We found that the number of matching frames is significantly reduced (up to 13 times fewer matching frames for some queries perturbed by adding irrelevant speech) when using the median as a threshold instead of the mean. The reason is that if we look at the percentage of 1's in the binary image averaged across all the reference frames, then this percentage is 8.9% for the mean and 50% for the median. Thus, *mean* eliminates most of the noise, while the *median* keeps most of the noise, resulting in many more poor matches.

Finally, each binary image is divided into small square tiles of size 11×11 resulting in 744 tiles[2]. The sum of all the 121 elements within each tile is computed to obtain a quantized image. Then, a few salient tiles that have the highest sums are selected. The fingerprint encodes the positions of these salient tiles and eliminates their sums. The example in Fig. 2 shows a binary image divided into 16 tiles $(D = 16)$ and 6 salient tiles $(d = 6)$ are selected. The positions of these 6 salient tiles represent the compact fingerprint.

Query fingerprints are created in the same way as the reference fingerprints. However, we noticed that many queries in TRECVID 2010 dataset are speeded up or slowed down compared to their corresponding references. Our fingerprints are sensitive to speed modifications. For this reason, we have produced two additional versions of query fingerprints with query audio speeded up or slowed down by 9% by increasing/decreasing the sampling frequency of the audio file (i.e. time-frequency scaling by resampling). Obviously, this operation does not solve the problem of speed difference between the queries and the references, but allows us to test the capability of our system to detect transformed queries with small speed variations. Note that the speed modification module is used only for TRECVID 2010 dataset. We present the performance of our system on TRECVID 2009 dataset without the additional speeded up or slowed down versions (TRECVID 2009 queries do not have any speed modifications). The principal intention of this work is to propose a fingerprinting system that is highly robust to audio transformations T1 through T7 shown in Section VII. Note that time and/or frequency scale modifications have a large impact on detection performance of many CBCD systems [14]. Different approaches have been proposed to handle these audio transformations [36], [38], [39].

[2]Because the 333×257 binary image is not divisible by 11, some regions are smaller than the rest of the equally sized regions.

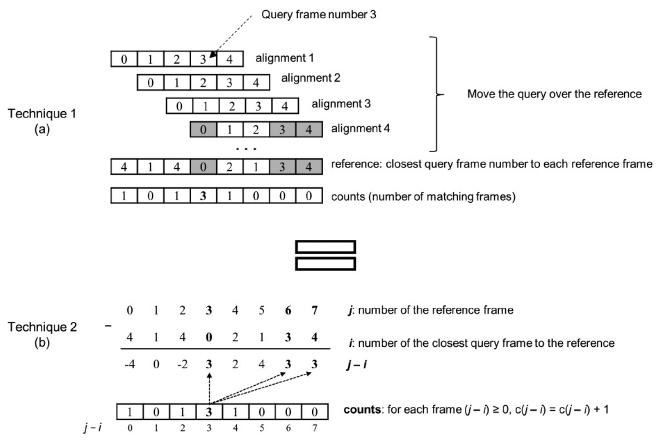
Query frame number 3
alignment 1
alignment 2
alignment 3
alignment 4
Move the query over the reference
Technique 1 (a)
reference: closest query frame number to each reference frame
counts (number of matching frames)
Technique 2 (b)
j: number of the reference frame
i: number of the closest query frame to the reference
j − i
counts: for each frame (j − i) ≥ 0, c(j − i) = c(j − i) + 1

Fig. 3. Matching frames based on nearest neighbor search.

B. Retrieval

Once all the reference and query fingerprints have been created, a search is performed to see if the query is a copy of an original audio in the reference database. Two principal steps for the retrieval process are:

Similarity search: Each reference frame is labeled with the frame number of its closest query fingerprint. The query consists of a number of frames, say *n*. For each query frame we compute its fingerprint as shown in Fig. 2. Similarly, each reference consists of a number of frames *m*, with a fingerprint extracted for each frame as shown in Fig. 2. For the fingerprint corresponding to each reference frame, we find the closest query fingerprint. The query frame number of this closest query fingerprint is then associated with that reference frame. In other words, in similarity search, we are associating a query frame number to each reference frame through this search. In this similarity search, the similarity measure is equal to the number of *salient positions* that coincide (i.e. *intersection* between reference and query fingerprint elements). Each reference frame is then labeled with the frame number of the closest query fingerprint. The total number of frame matches in the similarity search between a query (containing *n* frames) and a reference (containing *m* frames) is $n \times m$ (the total computing is proportional to $n \times m \times d$).

Matching: After the closest query frame has been found for each reference frame, the total number of reference frames that match the query frame-synchronously is computed: We move the query over the reference, and we count the number of reference frames that match exactly the query frame number for each alignment [see Fig. 3(a)). This count represents the confidence in the match between the query and the reference (i.e. the score). In Fig. 3(a), the reference fingerprints are labeled with the frame number of the closest query fingerprint (this operation is performed in the previous step: similarity search). The row labeled "counts" shows the total number of matching frames for each alignment of the query to the reference. Then, the best segment match is found by looking for the reference frame with the highest count. In the example of Fig. 3(a), the best count is equal to 3 and is obtained with *alignment 4* (matched frames between the query and the reference are highlighted with grey background).

A faster algorithm to perform this same operation is illustrated in Fig. 3(b). The counts of matching frames using this algorithm are obtained as follows: for each reference frame *j*, we increment the count c(*j* – *i*) as follow:

$$c(j-i) = c(j-i) + 1 \qquad (1)$$

where:

j	is the frame number of the reference.
i	is the label of frame j (i.e. the frame number of the closest query fingerprint is i).
$(j-i)$	is the index of the vector labeled *counts*.
$c(j-i)$	corresponds to the contents of the vector labeled *counts* at position $(j - i)$.

In Fig. 3(b), the best matching segment is obtained when the first query frame is aligned with the reference frame number 3 and results in 3 matching frames. These 3 frames are: $(j = 3, i = 0), (j = 6, i = 3)$ and $(j = 7, i = 4)$.

As we can see from Fig. 3(b), the total computing for the *matching* step is proportional to the total number of reference frames *m*, while the computing for the *similarity* search is proportional to $n \times m \times d$. In fact, the *matching* step is very fast and the total search time is dictated by the similarity search time. Therefore, in the next two sections, we propose two techniques to accelerate the similarity search step: (1) parallelize the algorithm using a Graphics Processing Unit (GPU) and (2) perform clustering on the reference fingerprints. The two techniques together reduce the computing time significantly.

IV. GPU-Based Search

As stated in the previous section, we accelerate the similarity search using a GPU. We introduced in [6] a detailed description of an efficient parallel design on GPU for this similarity search. We proposed two GPU implementations: sorting-based and hashing-based algorithms.

Our experiments showed that the sorting-based algorithm is slower than hashing-based algorithm (although it accelerates the search by 379 compared to its CPU implementation). We briefly describe in this section the design of the hashing-based algorithm as implemented on the GPU.

The fingerprint described in the previous section encodes the *positions* of salient regions selected from the binary image. The similarity between two fingerprints is defined as the intersection between the elements of these two fingerprints. For example, if *F1 = {1, 3, 4, 6, 8}* and *F2 ={1, 2, 4, 7, 9}*, then *similarity (F1, F2) = count {1, 4} = 2*. Formally, for two fingerprints F1 and F2, the similarity between them is:

Algorithm 1 Hashing-based similarity computation

```
Input: Two vectors $v1$ and $v2$ of size $d$
Output: The similarity between $v1$ and $v2$
 1: $sim = 0$
 2: create a vector $hash$ of size $D$
 3: for $i = 1$ to $D$ do
 4:     $hash[i] = 0$
 5: end for
 6: for $i = 1$ to $d$ do
 7:     $hash[v1[i]] = 1$
 8: end for
 9: for $i = 1$ to $d$, do
10:     $sim = sim + hash[v2[i]]$
11: end for
12: return $sim$
```

$$\mathrm{F1} \cap \mathrm{F2} = \mathrm{count}\,\{\mathrm{x} : \mathrm{x} \in \mathrm{F1and}x \in \mathrm{F2}\} \qquad (2)$$

To compute the similarity given by (2), the hashing-based algorithm (Algorithm 1) converts one *d*-dimensional fingerprint (vector *v1*) into a vector of *D* dimensions $(d << D)$, and then looks for matching entries of the second *d*-dimensional fingerprint in this *D*-dimensional vector (See Fig. 2 for definition of *d* and *D*). This algorithm has a linear time complexity. However, memory is a critical commodity on a GPU, and a *D*-dimensional vector, for every thread, may drain GPU's resources (in our experiments $D = 744$ compared to a maximum $d = 44$). We show in Section VII.C that the GPU memory space used to store the *D*-dimensional vector has a significant impact on performance.

The first step in any GPU implementation is to transfer data from the host (CPU memory) to the device (GPU memories). Since the GPU has limited memory space, we process the reference set by portion. We transfer a reference portion to the GPU's global memory, and we perform the similarity search on this portion, then we process the next portion until all reference fingerprints have been processed. The number of query fingerprints is small and all the query fingerprints are transferred to the global memory. Once query and reference fingerprints are transferred into global memory, the GPU launches a kernel that performs the parallel portion of the application. In this kernel, hundreds of threads are executed in parallel, where each of them finds the frame number of the closest query fingerprint for one reference fingerprint.

Algorithm 1a presents the proposed kernel, where the input are the reference and query fingerprints (loaded into global memory) and the output is a vector containing the frame number of the closest query fingerprint for each reference fingerprint. This similarity search uses the hashing-based algorithm. Each thread in this kernel starts by loading one *d*-dimensional reference fingerprint into registers or local memory (depending on dimension *d*: if there are not enough registers, then local memory is used to store one reference fingerprint for each thread). A *D*-dimensional vector (named *hash*) is created in shared memory to hash the elements of the query. It is also possible to switch the places for storing these variables by using

Algorithm 1a GPU kernel of hashing-based similarity search

```
Input: reference fingerprints, query fingerprints
Output: the closest query frame to each reference frame
 1: create a vector hash of size $D$ in shared memory
 2: create a vector ref of size $d$ in registers/local memory
 3: for each thread do
 4:     load in ref one $d$-dimensional reference frame from
        global memory
 5:     $max = 0$
 6:     $sim = 0$
 7:     $pos = -1$
 8:     for each query frame $n$ in global memory do
 9:         for $i = 1$ to $D$ do
10:             $hash[i] = 0$
11:         end for
12:         for $i = 1$ to $d$ do
13:             $hash[queryframe[n][i]] = 1$
14:         end for
15:         synchronize threads
16:         for $i = 1$ to $d$ do
17:             $sim = sim + hash[ref[i]]$
18:         end for
19:         if ($sim > max$) then
20:             $max = sim$
21:             $pos = n$
22:         end if
23:     end for
24: end for
25: $results$ [threadId] = $pos$
26: return $results$
```

Algorithm 2 Similarity computation without hashing

```
Input: Two vectors $v1$ and $v2$ of size $d$
Output: The similarity between $v1$ and $v2$
1: $sim = 0$
2: for $i = 1$ to $d$ do
3:     for $j = 1$ to $d$ do
4:         if ($v1(i) == v2(j)$) then
5:             $sim = sim + 1$
6:         end if
7:     end for
8: end for
9: return $sim$
```

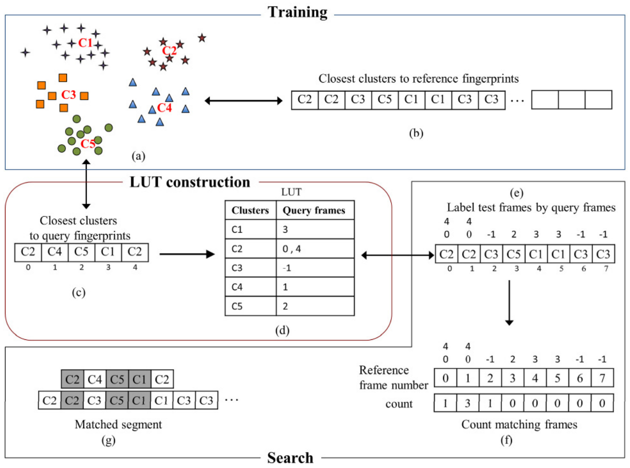

Fig. 4. Main components of the clustering-based technique.

the *hash* vector to hash the reference instead of the query. In this case, the D-dimensional vector will be created in the local memory instead of the shared memory, and a d-dimensional vector will store the query in shared memory. We explore in Section VII.C these two possibilities, and we show that the place where the D-dimensional vector is stored impacts significantly the speed of the algorithm. Note that initializing the *hash* vector (lines 9-11) and hashing the query (lines 12-14) are performed by all the threads within each thread block (since the *hash* vector is stored into shared memory making it accessible to all the threads in the same block). However, we load only one query fingerprint into shared memory, instead of partitioning the data into several portions where each portion contains as many fingerprints as possible that can fit into the shared memory. We show in Section VII.C that our strategy of loading only one query fingerprint into shared memory and processing query frame-by-frame results in better performance.

We also tried another similarity search algorithm that does not require the d-dimensional vector to be expanded into a D- dimensional vector (see Algorithm 2). However, its computational complexity is $d \times d$, instead of d, but requires significantly less memory. We tested Algorithm 2 with $d = 44$ on 10 queries, and it turned out to be 20 times slower (93242 seconds compared to 4449 seconds for Algorithm 1). It is important to optimize both memory and computing on the GPU. The algorithm also slows down due to the conditional if statement (thread divergence serialization).

V. Clustering-Based Technique

Here, we describe an efficient similarity search that further reduces the run time. Fig. 4 shows the main components of the proposed technique. First, we partition the reference fingerprints into different clusters and assign the closest cluster to each reference fingerprint. Second, we construct a lookup table based on the query fingerprints and the reference clusters. Finally, we compute the number of matching frames between the query and the reference. The details are:

A. Training

The idea is to partition the millions of reference fingerprints into smaller groups of similar fingerprints, and to represent each group by one representative fingerprint. A clustering algorithm partitions the reference fingerprint space into k separate clusters. The centroid of each cluster represents all the fingerprints in that cluster. This clustering reduces the number of fingerprint comparisons from the number of reference fingerprints (tens of millions) to the number of cluster centroids (tens of thousands).

In our experiments, we used a k-means like clustering on the reference fingerprints (Fig. 4(a)). Our k-means implementation differs from the original algorithm as follows:

1) The similarity function is defined as the intersection between two data point elements, since fingerprints encode positions of salient regions (see eq. (2)).
2) The elements of each centroid are defined by looking for the top-d positions shared by all the fingerprints in the cluster, where d is the fingerprint dimension. For example, if there are four fingerprints $P_1 = \{1, 2, 3, 5, 6\}$, $P_2 = \{2, 3, 5, 6, 9\}$, $P_3 = \{2, 5, 6, 11, 15\}$ and $P_4 = \{3, 7, 8, 10, 15\}$ in the cluster C, and $d = 5$, then the new centroid of this cluster C is $= \{2, 3, 5, 6, 15\}$.

After grouping all the reference fingerprints into k separate clusters, we label each reference fingerprint with the cluster number it belongs to (Fig. 4(b)). In short, we perform k-means clustering on all the reference fingerprints, and return k centroids and a vector that contains for each reference frame the cluster number to which it belongs. This step is performed offline only once for a reference fingerprints database, and this clustering runtime does not affect the query search time.

B. Lookup Table Construction

For each query, we compute the similarity between each query fingerprint and all the centroids generated in the clustering step. We label each query fingerprint with its closest centroid (Fig. 4(c)). In this scenario, two similar query and reference fingerprints have a high likelihood of belonging to the same cluster, even if their fingerprints are not identical.

To accelerate the search, we construct a lookup table (LUT) where the table index is the cluster number, and the output is the query frame numbers that belong to this cluster (Fig. 4(d)). Note that a -1 value in the LUT means that there is no query fingerprint closest to this cluster (e.g. C3 in Fig. 4(d)). Each reference frame is then assigned the frame number of the nearest query fingerprint using this lookup table.

C. Matching Algorithm

The matching algorithm is similar to that shown in Fig. 3, except that we compute the closest query frame to each reference cluster center instead of to each reference frame. This modification reduces the computing significantly. The matching between the query frames and the reference cluster centers is facilitated by the lookup table (LUT) shown in Fig. 4(d). For each reference frame, we find the query frame numbers that match the corresponding reference frame cluster label using the LUT (Fig. 4(e)). In other words, we label each reference frame by the query frame numbers that have the same cluster label as the reference frame cluster label. Then we compute the number of frame-synchronous frame matches (between the query and the reference) by updating the count $c(j-i) = c(j-i)+1$ for each frame j of the reference with label i, where $i > -1$(i is the query frame number from the LUT corresponding to the cluster Id of frame j of the reference). An illustration of this search is shown in Fig. 4(f) where the best count is 3 when the first query frame is overlaid on the reference starting with the 2nd frame. Fig. 4(g) shows this alignment with matching frames in a grey background.

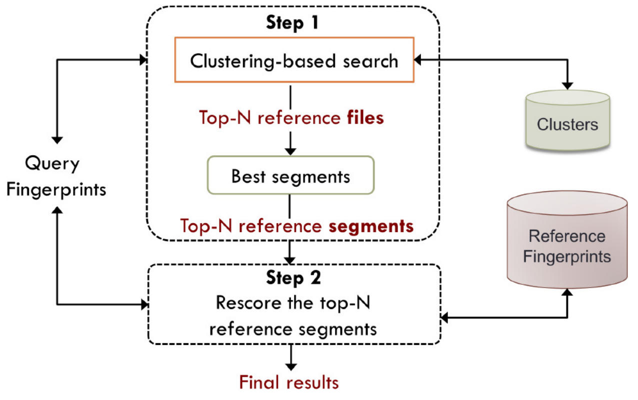

Fig. 5. Illustration of the two-step search. The Top N results generated from step 1 are rescored in step 2 for accurate results.

VI. Two-Step Search

The clustering-based technique as described above reduces the similarity search time by more than hundred. However, the system misses many queries that are detected with high scores when we do not use clustering. In fact, several transformations (especially those that add irrelevant speech to the query) make query fingerprints very different from their corresponding reference fingerprints, and therefore belong to different clusters. This mismatch decreases the number of matching frames, since a query and a reference fingerprint match only when they belong to the same cluster. In the original search (without clustering), the number of matching frames is computed after labeling each reference frame with the closest query frame (Fig. 3). Using the closest frame (instead of closest cluster) as a metric between query and reference fingerprints increases the likelihood of matching fingerprints from audio transformed with added speech. Thus, we propose to perform the search in two steps as shown in Fig. 5. The first step uses the clustering-based technique to compute and rank the scores of all reference files as shown in Fig. 4. Then, we keep the top-N reference files with the highest scores. For each of these top-N reference files we extract the best matching segment (i.e. segment that has the highest score). The second step rescores these top-N reference segments without clustering as in Fig. 3, using the nearest neighbor search between the query and reference fingerprints. This two-step technique reduces run time significantly while having minimal effect on min NDCR (see Section VII.D).

VII. Results and Analysis

This section evaluates our system on the well-known TRECVID 2010 CBCD dataset. First, we present results of our GPU implementation. Then, we use the GPU implementation to optimize the run time versus performance for the clustering-based technique. Finally, we compare the proposed system to the *NN-based* [21], *MASK* [11], the implementation of the *energy difference* fingerprint [8] and *Shazam* [13] (we used the

TABLE I
AUDIO QUERY TRANSFORMATIONS

Transformation	Description
T1	Nothing
T2	mp3 compression
T3	mp3 compression and multiband companding
T4	bandwidth limit and single band companding
T5	mix with speech
T6	mix with speech, then multiband compress
T7	bandpass filter, mix with speech, compress

implementation of Shazam by Dan Ellis [40] with the default parameters. We refer to it as *Ellis' Shazam-based* system). We also validate the performance of our system on TRECVID 2009 dataset, and we compare our results to the *WASF* audio features used in [17] and the *coherency* vocabulary method proposed in [31].

A. Datasets

TRECVID 2009 and 2010 are two datasets provided by NIST [41] for the task of video copy detection. TRECVID 2010 dataset consists of a reference collection of more than 11 000 videos for a total of 400 hours of videos, whereas TRECVID 2009 contains 385 hours of reference videos. For each dataset, there are 201 original queries, each query altered with one of the 7 audio transformations described in Table I for a total of 1407 transformed audio queries. The length of queries vary from 3 seconds to 3 minutes and a query can be one of these 3 types: (1) transformed fragment of a reference video; (2) contains a transformed fragment of a reference video; (3) fragment of a video not in the reference video database. See [42] for the query creation framework.

B. Evaluation Metrics

To evaluate the accuracy of locating a copied fragment within a video, we use Mean F1 that is defined as the harmonic mean of precision and recall. For copy detection accuracy, we use the minimal Normalized Detection Cost Rate (min NDCR). *NDCR* is a weighted cost combination of the probability of missing a true copy (P_{Miss}) and the false alarm rate (R_{FA}), computed as:

$$\begin{aligned} NDCR &= P_{Miss} + \beta^{*} R_{FA} \\ \text{with} \quad P_{Miss} &= \mathrm{FN}/\mathrm{N}_{\mathrm{target}} \\ \beta &= \mathrm{C}_{\mathrm{FA}}/(\mathrm{C}_{\mathrm{Miss}}{}^{*}\mathrm{R}_{\mathrm{target}}) \\ R_{FA} &= \mathrm{FP}/(\mathrm{T}_{\mathrm{refdata}}{}^{*}\mathrm{T}_{\mathrm{query}}) \end{aligned} \tag{3}$$

where $\mathrm{C}_{\mathrm{Miss}}$ and C_{FA} are the costs of an individual miss and an individual false alarm, respectively; FN and FP are false negative and false positive; $\mathrm{T}_{\mathrm{refdata}}$ is the total length (in hours) of the entire reference dataset, and $\mathrm{T}_{\mathrm{query}}$ is the total length (in hours) of all the queries for a transformation; $\mathrm{N}_{\mathrm{target}}$ is the total number of copies and $\mathrm{R}_{\mathrm{target}}$ is the a priori target rate for the application of interest.

In the TRECVID CBCD evaluation, above parameters were defined for two different application profiles: "balanced" and "no false alarm" (NOFA) profile. In NOFA profile, which is more difficult, the cost of an individual false alarm is 1000 times the cost of a missed query, while in balanced profile both missed and false alarms are assigned a cost of 1. We report results using the NOFA profile here with::

$$\mathrm{R}_{\mathrm{target}} = 0.005/\mathrm{hr}^2,\ \mathrm{C}_{\mathrm{Miss}} = 1 \,\mathrm{and}\, \mathrm{C}_{\mathrm{FA}} = 1000 \tag{4}$$

Many reference audio files in TRECVID 2010 dataset have duplicates that skew the results. We have removed these duplicate audio files before evaluating any systems we tested. The duplicate reference files are (removed files marked in italic): *102*/50, *10907*/5225, *2002*/2062, *3747*/*4328*/9877, *10236*/*11449*/7414, *5680*/7130, *2572*/*2552*/2539, *3725*/5716.

TABLE II
HASHING-BASED RUN TIMES (IN SEC) WITH DIFFERENT CONFIGURATIONS

D	12	24	44
Configuration 1: $k = 256$	1037	2321	15405
Configuration 2: $k = 1$ (reference)	1143	2477	5030
Configuration 3: $k = 1$ (query)	444	915	4449
CPU	66870	132564	253148

C. GPU Implementation Performance

We run the CPU-based implementation on an Intel(R) Xeon(R) CPU at 2.3 GHz. For the GPU implementation we use the GPU GeForce GTX 580 (based on NVIDIA's Fermi architecture). This GPU has 16 SMPs; each one has 32 cores for a total of 512 cores. GTX 580 has multiple memory spaces: global memory (1.5 GB), shared memory (48 KB/SMP shared by all the blocks in the SMP), 32-bit registers (32 k registers per SMP) and local memory (512 KB/thread).

In the first part of our experiment we evaluate the run time of our system using all the reference collection and only 10 queries of varying length. Later on, we present the best GPU implementation results on all the 1407 queries.

Table II compares run times of the GPU implementation of hashing-based algorithm with 3 different configurations and its CPU implementation (last line of Table II). These configurations differ in how the shared memory is used to load the query and reference frames for computing the similarity measure shown in Algorithm 1. In the first configuration we load $k = 256$ query fingerprints into Shared Memory (SM), whereas with the two other configurations only one fingerprint $(k = 1)$ is loaded into SM. The difference between configurations 2 and 3 (with $k = 1$) is the place where the *D*-dimensional vector is stored (see Algorithm 1a). In configuration 2, the reference fingerprint is hashed into a *D*-dimensional vector in the local memory, whereas in configuration 3 we hash the query fingerprint into a *D*-dimensional vector in the SM.

From Table II we see that loading only *one* query fingerprint to SM (configuration 2) instead of 256 (configuration 1) is more efficient when $d = 44$ (3 times faster). Because the improved configuration loads only one query fingerprint to the SM, we have enough SM space to run many more threads concurrently. However, configuration 2 with $d = 12$ and with $d = 24$ are

TABLE III
SYSTEM PERFORMANCES USING VARYING DIMENSIONS WHEN TESTED WITH ALL THE QUERIES ON TRECVID 2010

	12	24	44
Min NDCR	0.149	0.135	0.129
Mean F1	0.903	0.9	0.885
Times (s)	46	95	438

TABLE IV
NUMBER OF MISSED QUERIES FOR SR-44 AND CSR-44

Method	T1	T2	T3	T4	T5	T6	T7	Total
SR-44	10	10	11	12	21	20	20	104
CSR-44	12	14	28	27	27	44	42	194
CSR-44, *N*=2	12	13	22	20	25	39	36	167

TABLE V
MIN NDCR FOR SR-44 AND CSR-44

Method	T1	T2	T3	T4	T5	T6	T7	Average
SR-44	0.09	0.09	0.112	0.097	0.172	0.157	0.187	0.129
CSR-44	1.000	1.000	1.000	1.000	0.448	0.679	0.709	0.833
CSR-44, *N*=2	0.09	0.097	0.164	0.149	0.194	0.291	0.269	0.179

slightly slower. This is corrected if we hash the query fingerprint into a *D*-dimensional vector (configuration 3) instead of the reference (configuration 2), leading to a two-fold reduction in computing time. This shows that hashing the *D*-dimensional query vector into SM instead of hashing reference vector into local memory works well. Note that, query has to be hashed in the SM so that all the threads in the block can see it. Hashing query in local memory would lead to each thread hashing the query vector. The reference vector cannot be hashed in the SM as there will be too many reference frames to hash (as many as number of active threads).

The last line of Table II shows that compared to the CPU implementation, the best GPU implementation (configuration 3) speeded up the hashing-based algorithm by 150, 144 and 56 times for *d* equal to 12, 24 and 44, respectively.

Table III shows the min NDCR, Mean F1 and run time averaged over all transformations (the run time is averaged over all queries) when processing all the 1407 query files using hashing-based algorithm with the best configuration (i.e. configuration 3). Note that most of the search time is for the similarity search, the matching module takes less than 2 seconds (regardless of *d*).

D. Clustering-Based Search Performance

In this section, we compare the clustering-based similarity search (CSR) with salient regions search (SR, no clustering) for computing and performance tradeoffs. The number of missed queries and the min NDCR for each transformation for SR-44 and CSR-44 (dimension $d = 44$) are shown with grey background in Tables IV and V, respectively. In these tables, the search with CSR-44 is without rescoring the top *N* results (i.e. no two-step search).

The average run time per query is equal to 3 seconds for CSR-44 compared to 438 seconds for SR-44. Although CSR-44 reduces the run time significantly, it misses more queries than SR-44. Furthermore, scores of many imposter queries are very high in CSR-44, resulting in high decision thresholds (threshold that separates true positives from false alarms) leading to a high min NDCR (Table V). For example, the min NDCR for transform T1 increases from 0.09 for SR-44 to 1 for CSR-44, although CSR-44 missed only 2 more queries than SR-44 (Table IV).

To improve the detection performance, we process the query in two steps as described in Section VI. The number of missed queries and the min NDCR for CSR-44 with the top 2 choices rescored $(N = 2)$ is shown in the last row of Tables IV and V, respectively. From Table IV, we can see that the two-step search reduces the number of missed queries for almost all the transformations and results in overall 27 less missed queries. The min NDCR for all the transformations are reduced significantly (see Table V), and the min NDCR are correlated with the number of missed queries, indicating a good decision threshold. In addition, $N = 2$ does not increase the run time, and CSR-44 is still 146 times faster than SR-44.

Fig. 6 shows the CSR performance tradeoff with increasing *N*, for $d = 12$ and $d = 44$ dimensions. This figure also compares the CSR performance for rescoring the top *N files* versus rescoring the top *N segments*. Fig. 6(a) shows the min NDCR for each transformation with $d = 44$ and $N = 50$ files for varying number of clusters. From this figure, we see that using 1000 clusters gave the highest min NDCR (worst results) for almost all the transformations. Results of min NDCR obtained with 10 thousand and 20 thousand clusters are very close. The average min NDCR over all transformations is equal to 0.147 when using 10 thousand clusters compared to 0.148 when using 20 thousand clusters. Since 10 thousand clusters gave the best results, the rest of the experiments are performed using these 10 thousand clusters.

We see a significant run time reduction when the top *N segments* are rescored instead of the top *N files* for $d = 12$ (Fig. 6(e)) and $d = 44$ (Fig. 6(b)). The run time for $d = 44$ increases by only 6 seconds from $N = 10$ segments to $N = 500$ segments compared to 31 seconds when the number of files increase from 10 to 500. Also, the run time difference between $d = 12$ and $d = 44$ becomes less important when rescoring segments instead of files. In fact, the run time is almost identical for both the dimensions when $N < 300$ segments. The clustering-based similarity search reduces the run time difference between $d = 12$ and $d = 44$. Without clustering, the search algorithm becomes 10 times slower when *d* increases from 12 to 44 (Table III).

From Fig. 6(d) we see that the number of missed queries decrease as *N* increases for $d = 44$. However, the top *N* files result in fewer missed queries than top *N* segments (similar result is obtained with $d = 12$ as shown in Fig. 6(g)). The same can also be said for min NDCR (averaged over all transforms) for $d = 12$ (Fig. 6(f)) and $d = 44$ (Fig. 6(c)). However for some cases, the min NDCR becomes higher with increasing N, even though more queries are correctly detected. For example, with $N = 200$ files, the system missed 127 queries and gave a min NDCR of 0.140 (Fig. 6(c) and Fig. 6(d), respectively]. When *N* is increased to 500 files the number of missed queries is

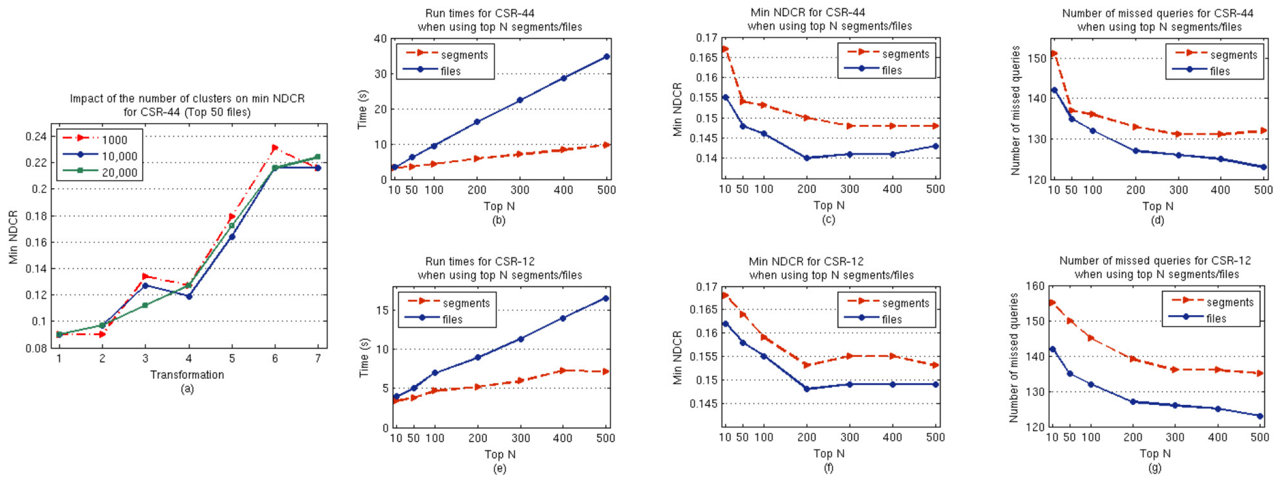

Fig. 6. Performance of the clustering-based technique: (a) impact of the number of clusters on min NDCR; (b), (c) and (d) evolution of run time, min NDCR and missed queries, respectively, for CSR-44 with Top N files and segments ((e), (f) and (g) for CSR-12).

TABLE VI
MIN NDCR FOR NN-BASED, ELLIS' SHAZAM-BASED, SR AND THE BEST RESULTS OF CSR USING 12 AND 44 DIMENSIONS

Method	**T1**	**T2**	**T3**	**T4**	**T5**	**T6**	**T7**	**Average min NDCR**	**Average times (s)**
SR-12	0.104	0.112	0.149	0.112	0.179	0.187	0.201	0.149	46
CSR-12 Top 200 files	0.097	0.097	0.134	0.112	0.179	0.209	0.209	0.148	9
CSR-12 Top 200 segments	0.097	0.104	0.149	0.119	0.187	0.209	0.209	0.153	5
SR-44	0.09	0.09	0.112	0.097	0.172	0.157	0.187	0.129	438
CSR-44 Top 200 files	0.09	0.097	0.097	0.119	0.164	0.209	0.201	0.140	16
CSR-44 Top 300 segments	0.09	0.097	0.127	0.127	0.172	0.209	0.216	0.148	7
NN-based	0.179	0.187	0.194	0.187	0.201	0.194	0.209	0.193	178
Ellis' Shazam-based	0.373	0.366	0.507	0.47	0.515	0.597	0.59	0.488	1.5

reduced to 123, yet the min NDCR increases to 0.143. This can be explained by the fact that with increasing N, the system not only detects correctly more queries, but may also give high scores for some imposters, resulting in a high decision threshold. Thus, the optimal number N corresponds to the result with the lowest min NDCR. For $d = 44$ the best results are achieved with $N = 200$ files or $N = 300$ segments (Fig. 6(c)). The lowest min NDCR for $d = 12$ is obtained with $N = 200$ files or segments (Fig. 6(f)).

Table VI compares min NDCR for each audio transformation for SR versus CSR (rows with grey background) using both 12 dimensions ($N = 200$ files or segments) and 44 dimensions ($N = 200$ files or 300 segments). The last two rows of Table VI show the min NDCR of NN-based [21] and Ellis' Shazam-based [13] systems. From Table VI we see that SR-44 gives the lowest min NDCR averaged over all transformations. Using clustering increases the min NDCR from 0.129 (for SR-44) to 0.140 (for CSR-44 with top 200 files); however, CSR-44 speeds up the search by 27 times. The clustering-based search seems to work better for $d = 12$ and $N = 200$ files, reducing the min NDCR from 0.149 to 0.148 (compared to SR-12) while being 5 times faster. On the other hand, using the top N segments instead of files increases the min NDCR for most of the transformations. However, it speeds up the search by 9 and 62 times ($d = 12$ and $d = 44$, respectively) compared to SR-12 and SR-44, respectively.

When we compare our system to the two other systems we find that SR-44 achieves the lowest min NDCR for all the transformations. In fact, the min NDCR averaged over all transformations goes down from 0.193 for NN-based system to 0.129 for SR-44. Ellis' Shazam-based system gave the worst results with a min NDCR averaged over all transformations of 0.488. When we look at Ellis' Shazam-based system results, we find that a large number of missed queries are queries of type (2) (reference video embedded in a non-reference video). However, Ellis' Shazam-based system is the fastest system and requires less than 2 seconds, on average, to process a query. NN-based system takes roughly 178 seconds which makes it 4 times slower than SR-12. Although CSR reduces the detection performance compared to SR, it still gave lower min NDCR compared to NN-based and Ellis' Shazam-based systems. CSR-44 with $N = 2$ gave better detection performance (min NDCR $= 0.179$) than these two systems, and is 60 times faster than the NN-based system.

TABLE VII
MIN NDCR FOR SR-44 AND CSR-44 COMPARED TO WASF AND COHERENCY ON TRECVID 2009 DATASET

Method	T1	T2	T3	T4	T5	T6	T7	Average min NDCR	Average times (s)
SR-44	0.06	0.067	0.082	0.075	0.09	0.157	0.127	0.094	214
CSR-44 Top 300 segments	0.082	0.09	0.119	0.097	0.119	0.194	0.224	0.132	3
WASF	0.090	0.09	0.09	0.09	0.194	0.172	0.187	0.130	54
coherency	0.142	0.306	0.629	0.485	0.530	0.697	0.918	0.529	8

Besides the NN-based and Ellis' Shazam-based systems, our system outperforms other systems such as MASK [11] and the implementation of the energy difference fingerprint [8]. On TRECVID 2010 dataset, MASK achieved a min NDCR averaged over all transformations of 0.43 compared to 0.55 achieved by the energy difference fingerprint (these results are published in [11]). Even CSR-12 with Top 200 segments (see Table VI) gave a significantly lower min NDCR of 0.153.

E. Validation Results on TRECVID 2009

In order to validate the copy detection results obtained on TRECVID 2010 dataset, we tested our system on TRECVID 2009 dataset with the same parameters. For the TRECVID 2009, we do not generate additional fingerprints with varying speeds. Table VII shows min NDCR for SR-44, CSR-44 (with Top 300 segments), WASF [17] and for the coherency vocabulary method [31]. SR-44 shows good detection results and achieves the lowest min NDCR for all the transformations. Although the averaged min NDCR of CSR-44 is 0.132 compared to 0.094 for SR-44, CSR-44 speeded up the search by 71 times compared to SR-44. In addition, CSR-44 achieves comparable performance to WASF while being 18 times faster. The coherency vocabulary method is 7 times faster than WASF, but results in relatively poor detection performance (averaged min NDCR of 0.529).

F. Scalability

To test the detection performance of our system on a larger reference dataset, we doubled the reference dataset by combining TRECVID 2009 and 2010 reference datasets. The evaluation of our system using TRECVID 2010 queries shows that doubling the size of the reference dataset slightly impacts the detection performance of our system (average min NDCR of 0.130 with the new dataset compared to 0.129 achieved on TRECVID 2010 references only for SR-44). Similarly, the average min NDCR of CSR-44 with Top 300 segments goes from 0.148 using TRECVID 2010 only, to 0.150 with the doubled dataset.

In addition, Table VIII compares the run times of our system on the doubled reference dataset when using two different GPUs: *GTX 580* and *GTX Titan X*.

Table VIII shows that doubling the size of the reference dataset leads to a linear increase of the run time for SR-44 (from 438 secs to 872 secs) and sublinear increase for CSR-44 Top 300 segments (from 7 secs to 9 secs). GTX Titan X speeded up the search for SR-44 by almost 6 times compared to GTX 580. We also notice a small speed improvement for CSR-44 Top 300 segments when using *GTX Titan X*.

TABLE VIII
AVERAGE RUN TIMES (SECS) ON GTX 580 AND GTX TITAN X FOR SR-44 AND CSR-44 TOP 300 SEGMENTS

Device	SR-44	CSR-44 Top 300 segments
GTX 580	872	9
GTX Titan X	153	6

G. Ellis' Shazam-Based System Versus CSR-44

The question is how we can reduce the run time of our system to match the run time of Ellis' Shazam-based system. The average run time of CSR-44 with top $N = 2$ files is 3 seconds per query. With the two-step search, the matching algorithm is executed twice: the first time to count the number of matching *clusters* (first step), and the second time to count the number of matching *frames* (second step). Counting the number of matching clusters is slower than counting the number of matching frames for two reasons. First, in the second step of the retrieval algorithm we process only N reference files/segments compared to all reference files in the first step. Second, a reference frame can be labeled with multiple query frames (when several query fingerprints belong to the same cluster, see Fig. 4(e)) compared to only one query frame (i.e. the closest query frame to the reference) for the second matching step. In order to reduce the run time, we keep only one query frame when we perform the first matching algorithm instead of multiple frames. In the example of Fig. 4(d), this implies keeping only the query frame number 0 instead of 0 and 4 in the LUT where the input is C2.

When we tested the modified matching algorithm using CSR-44 with top 10 segments, our system became slightly faster than Ellis' Shazam-based system with an average run time of 1.3 seconds, which is also 137 times faster than the NN-based system. Understandably, the detection performance decreased, and CSR-44 gave an average min NDCR of 0.193 (same as NN-based system), which is far better than the min NDCR achieved by Ellis' Shazam-based system (0.488).

Note that the 1.3 second runtime includes the similarity search on GPU and two matching algorithms (first and second steps in Section VI). The matching algorithms are fast due to 2 reasons: The first matching algorithm in CSR-44 does not process many reference frames (i.e. there is -1 in the LUT). In addition, the second matching algorithm processes fewer reference frames: frames corresponding to only 10 reference segments.

On the other hand, we get better localization accuracy than the NN-based and Ellis' Shazam-based systems as shown in Table IX (higher F1 means better localization). Our method gave mean F1 averaged over all transformations of 0.911, compared to 0.807 and 0.693 for Ellis' Shazam-based and NN-based systems, respectively.

TABLE IX
MEAN F1 FOR THE PROPOSED SYSTEM (CSR-12 WITH TOP 200 SEGMENTS), NN-BASED AND ELLIS' SHAZAM-BASED SYSTEMS

System	T1	T2	T3	T4	T5	T6	T7	Average
CSR-12	0.923	0.935	0.925	0.923	0.892	0.898	0.883	0.911
NN-based	0.685	0.695	0.701	0.691	0.685	0.691	0.703	0.693
Ellis' Shazam-based	0.765	0.789	0.825	0.795	0.803	0.841	0.834	0.807

Note that the average run time of the CPU implementation of SR-44 is estimated to be more than 7 hours. The average run time of 1.3 seconds achieved by the proposed two-step search makes our algorithm 19472 times faster than the CPU implementation without clustering.

VII. CONCLUSION

In this work, we propose speed up of a state-of-the-art audio fingerprinting system appropriate for detecting audio copies subjected to complicated transformations. To generate audio fingerprints, this system converts the audio signal into 2-D binary images derived from the spectrogram. Each fingerprint encodes the positions of salient regions selected from this binary image. The similarity between two fingerprints is defined as the intersection between their elements.

Because of the high dimensionality of the fingerprints and the large volume of data, searching over millions of fingerprints is computationally challenging. We investigate the use of a GPU to reduce the computing: we describe an efficient way of utilizing the GPU memories to search for similar fingerprints in parallel on a database of 60 million fingerprints. Experimental results demonstrate that the GPU implementation can accelerate the search by up to 150 times compared to the CPU implementation.

To reduce the run time even further, we propose a two-step clustering based search. In the first step, we cluster the reference fingerprints into several thousand clusters, reducing the nearest-neighbor search time significantly. In the second step, we rescore the top N results obtained in the first step to produce more accurate copy detection. This two-step strategy works very well and achieves a speed up of 146 times while increasing the min NDCR from 0.129 to 0.179 on the TRECVID 2010 copy detection task. Basically, clustering creates a tradeoff between the speed and the detection accuracy. The experimental results show that using $d = 44$ and top 200 files accelerates the search 27 times and achieves a min NDCR of 0.140. For $d = 12$, the clustering-based technique not only speeds up the search, but also results in better detection performance (compared to SR-12 that does not use clustering-based search).

In addition, clustering allowed us to match the speed of Ellis' Shazam-based system on TRECVID 2010 with significantly better performance. Our system also outperformed MASK, WASF feature, Coherency vocabulary method, the energy difference fingerprints, and the robust NN-based system.

Future work will be mainly devoted to exploring new strategies to make the Salient-Regions audio fingerprint invariant to time-frequency scale modifications. A possible way to achieve this goal is to encode the positions of the selected salient regions relative to each other, instead of their positions within the window. A time-frequency modification applied to an audio signal leads to a proportional change in the time and frequency axes. This proposed strategy will ensure that the temporal and the spatial information will not be included in the fingerprint.

As another part of our future work, we will investigate other promising clustering algorithms for robustness and for computing reduction. Finally, we intend to adapt the proposed audio fingerprint extraction technique to the video copy detection problem. In this case, salient regions will be selected from video images instead of audio spectrograms.

REFERENCES

[1] Youtube. (Mar. 20, 2015). *YouTube Statistics* [Online]. Available: https://www.youtube.com/yt/press/statistics.html.

[2] Riaa. (Jun. 2015). *RIAA Statistics* [Online]. Available: https://www.riaa.com/physicalpiracy.php?content_selector=piracy-online-scope-of-the-problem.

[3] P. Cano *et al.*, "A review of algorithms for audio fingerprinting," in *Proc. IEEE Workshop Multimedia Signal Process.*, (Cat. No. 02TH8661), 2002, pp. 169–73.

[4] P. Cano *et al.*, "A review of audio fingerprinting," *J. VLSI Signal Process. Syst. Signal Image Video Technol.*, vol. 41, no. 3, pp. 271–284, 2005.

[5] C. Ouali, P. Dumouchel, and V. Gupta, "Efficient spectrogram-based binary image feature for audio copy detection," in *Proc. 40th IEEE Int. Conf. Acoust., Speech, Signal Process.*, Australia, 2015, pp. 1792–1796.

[6] C. Ouali, P. Dumouchel, and V. Gupta, "GPU implementation of an audio fingerprints similarity search algorithm," in *Proc. 13th Int. Workshop Content-Based Multimedia Index. (CBMI'15)*, Prague, Czech Republic, 2015.

[7] J. Lebosse, L. Brun, and J. C. Pailles, "A robust audio fingerprint extraction algorithm," in *Proc. 4th IASTED Int. Conf. Signal Process. Pattern Recognit. Appl.*, Anaheim, CA, USA, Feb. 14–16, 2007, pp. 269–74.

[8] J. Haitsma and T. Kalker, "A highly robust audio fingerprinting system," in *Proc. Int. Soc. Music Inf. Retrieval Conf. (ISMIR'02)*, 2002, pp. 107–115.

[9] W. Lezi *et al.*, "Contented-based large scale web audio copy detection," in *Proc. IEEE Int. Conf. Multimedia Expo. (ICME'12)*, 2012, pp. 961–966.

[10] A. Saracoglu *et al.*, "Content based copy detection with coarse audio-visual fingerprints," in *Proc. 7th Int. Workshop Content-Based Multimedia Index. (CBMI'09)*, 2009, pp. 213–18.

[11] X. Anguera, A. Garzon, and T. Adamek, "MASK: Robust local features for audio fingerprintingin *Proc. 13th IEEE Int. Conf. Multimedia Expo. (ICME'12)*, Melbourne, VIC, Australia, Jul. 9–Jul. 13, 2012, pp. 455–460.

[12] K. Yan, D. Hoiem, and R. Sukthankar, "Computer vision for music identification," in *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, 2005, pp. 597–604.

[13] A. L. C. Wang, "An industrial-strength audio search algorithm," in *Proc. SPIE Int. Soc. Opt. Eng.*, 2003, vol. 5307, no. 1, pp. 582–588.

[14] S. Baluja and M. Covell, "Waveprint: Efficient wavelet-based audio fingerprinting," *Pattern Recognit.*, vol. 41, no. 11, pp. 3467–3480, 2008.

[15] S. Baluja and M. Covell, "Audio fingerprinting: Combining computer vision data stream processing," in *Proc. IEEE Int. Conf. Acoust., Speech, Signal Process. (ICASSP'07)*, 2007, pp. 213–16.

[16] V. Chandrasekhar, M. Sharifi, and D. A. Ross, "Survey and evaluation of audio fingerprinting schemes for mobile query-by-example applications," in *Proc. Int. Soc. Music Inf. Retrieval Conf. (ISMIR'11)*, 2011, pp. 801–806.

[17] L. Mou *et al.*, "Content-based copy detection through multimodal feature representation and temporal pyramid matching," *ACM Trans. Multimedia Comput. Commun. Appl.*, vol. 10, no. 1, pp. 1–20, Art. no. 5, 2013.

[18] V. Garcia, E. Debreuve, and M. Barlaud, "Fast k nearest neighbor search using GPU," in *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit. Workshops (CVPRW'08)*, 2008, pp. 1–6.

[19] J. Pan and D. Manocha, "Fast GPU-based locality sensitive hashing for k-nearest neighbor computation," in *Proc. 19th ACM SIGSPATIAL Int. Conf. Adv. Geogr. Inf. Syst.*, 2011, pp. 211–220.

[20] H. Mohamed, H. Osipyan, and S. Marchand-Maillet, "Fast large-scale multimedia indexing and searching," in *Proc. 12th Int. Workshop Content-Based Multimedia Index. (CBMI'14)*, 2014, pp. 1–6.

[21] V. N. Gupta, G. Boulianne, and P. Cardinal, "CRIM's content-based audio copy detection system for TRECVID 2009," *Multimedia Tools Appl.*, vol. 60, no. 2, pp. 371–87, 2012.

[22] P. Cardinal, V. Gupta, and G. Boulianne, "Content-based advertisement detection," in *Proc. 11th Annu. Conf. Int. Speech Commun. Assoc.*, 2010, pp. 2214–2217.

[23] C. Ouali, P. Dumouchel, and V. Gupta, "A robust audio fingerprinting method for content-based copy detection," in *Proc. Int. Workshop Content-Based Multimedia Index. (CBMI'14)*, Austria, 2014.

[24] C. Ouali, P. Dumouchel, and V. Gupta, "Robust features for content-based audio copy detection," in *Proc. 15th Annu. Conf. Int. Speech Commun. Assoc.*, 2014.

[25] C.-C. Wang, J.-S. R. Jang, and W. Liou, "Speeding up audio fingerprinting over GPUs," in *Proc. Int. Conf. Audio Lang. Image Process. (ICALIP'14)*, 2014, pp. 5–10.

[26] J. Martinez *et al.*, "Fast parallel audio fingerprinting implementation in reconfigurable hardware and GPUs," in *Proc. 7th South. Conf. Programmable Logic (SPL'11)*, 2011, pp. 245–250.

[27] C. Bellettini and G. Mazzini, "A framework for robust audio fingerprinting," *J. Commun.*, vol. 5, no. 5, pp. 409–424, 2010.

[28] D. Sui, L. Ruan, and L. Xiao, "A two-level audio fingerprint retrieval algorithm for advertisement audio," in *Proc. 12th Int. Conf. Adv. Mobile Comput. Multimedia*, 2014, pp. 235–239.

[29] M. M. Esmaeili, M. Fatourechi, and R. K. Ward, "A robust and fast video copy detection system using content-based fingerprinting," *IEEE Trans. Inf. Forensics Secur.*, vol. 6, no. 1, pp. 213–226, Mar. 2011.

[30] H. Hermansky and N. Morgan, "RASTA processing of speech," *IEEE Trans. Speech Audio Process.*, vol. 2, no. 4, pp. 578–589, Oct. 1994.

[31] Y. Liu *et al.*, "Coherent bag-of audio words model for efficient large-scale video copy detection," in *Proc. ACM Int. Conf. Image Video Retrieval*, 2010, pp. 89–96.

[32] M. Douze *et al.*, "INRIA-LEARs video copy detection system," in *Proc. TREC Video Retrieval Eval. (TRECVID Workshop)*, 2008, pp. 1–8.

[33] E. Younessian *et al.*, "Telefonica research at TRECVID 2010 content-based copy detection," in *Proc. TREC Video Retrieval Eval. (TRECVID'10)*, 2010.

[34] J. Sivic and A. Zisserman, "Video google: A text retrieval approach to object matching in videos," in *Proc. 9th IEEE Int. Conf. Comput. Vis.*, 2003, pp. 1470–1477.

[35] M. Brookes. (Mar. 2011). *Voicebox: Speech Processing Toolbox for MATLAB* [Online]. Available: www.ee.ic.ac.uk/hp/staff/dmb/voicebox/voicebox.html.

[36] X. Zhang *et al.*, "SIFT-based local spectrogram image descriptor: A novel feature for robust music identification," *EURASIP J. Audio Speech Music Process.*, vol. 2015, no. 1, pp. 1–15, 2015.

[37] H. Jegou *et al.*, "BABAZ: A large scale audio search system for video copy detection," in *Proc. IEEE Int. Conf. Acoust., Speech, Signal Process. (ICASSP '12)*, 2012, pp. 2369–72.

[38] M. Malekesmaeili and R. K. Ward, "A local fingerprinting approach for audio copy detection," *Signal Process.*, vol. 98, pp. 308–321, 2014.

[39] M. Ramona and G. Peeters, "AudioPrint: An efficient audio fingerprint system based on a novel cost-less synchronization scheme," in *Proc. IEEE Int. Conf. Acoust., Speech, Signal Process. (ICASSP)*, 2013, pp. 818–822.

[40] D. Ellis. (May 2009). *Robust Landmark-Based Audio Fingerprinting*, vol. 4 [Online]. Available: HTTP: http://labrosa.ee.columbia.edu/~dpwe/resources/matlab/fingerprint

[41] G. Awad, P. Over, and W. Kraaij, "Content-based video copy detection benchmarking at TRECVID," *ACM Trans. Inf. Syst. (TOIS'14)*, vol. 32, no. 3, pp. 14, 2014.

[42] NIST. (Jun. 5, 2015). *Building Video Queries for Trecvid2008 Copy Detection Task*. [Online]. Available: http://www-nlpir.nist.gov/projects/tv2010/TrecVid2008CopyQueries.pdf.

Chahid Ouali (M'15) received the B.Sc. degree in computer science from the University of Sfax, Sfax, Tunisia, and the M.Ing. degree from the École de Technologie supérieure, Montréal, QC, Canada. He is currently pursuing the Ph.D. degree at the Department of Software and IT Engineering, École de Technologie Supérieure. He is also with the Centre de recherche informatique de Montréal (CRIM), Canada. His research interests include pattern recognition, information retrieval, and audio signal processing. He is working on audio and video fingerprinting methods for content-based copy detection.

Pierre Dumouchel (M'97) received the B.Eng. degree from McGill University, Montréal, QC, Canada, and the M.Sc. and Ph.D. degrees from the INRS—Télécommunications, Quebec City, QC, Canada. He has more than 25 years of experience in the field of speech recognition, speaker recognition, and emotion detection. He is the Director General at the École de technologie supérieure (ETS) of Université du Québec, Quebec City, QC, Canada.

Vishwa Gupta (LSM'14) received the B.Tech. degree from the Indian Institute of Technology (IIT) Kharagpur, Kharagpur, India, and the M.Sc. and Ph.D. degrees in electrical and computer engineering from Clemson University, Clemson, SC, USA. At Nortel, he worked on many speech recognition applications including ADAS Plus for directory assistance automation. At SpeechWorks, he was involved in research and development of applications as a Member of the Product Team. At IBM, he worked on their speech recognition engine to incorporate state-of-the-art algorithms and features. At the Centre de Recherche Informatique de Montreal (CRIM), he has been active in speech recognition, speaker diarization, keyword spotting, content-based audio/video copy detection, and automated advertisement detection.