This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TMM.2015.2389628, IEEE Transactions on Multimedia

Cloud-based Multimedia Content Protection System

Mohamed Hefeeda, *Senior Member, IEEE,* Tarek ElGamal, Kiana Calagari, and Ahmed Abdelsadek

Abstract

We propose a new design for large-scale multimedia content protection systems. Our design leverages cloud infrastructures to provide cost efficiency, rapid deployment, scalability, and elasticity to accommodate varying workloads. The proposed system can be used to protect different multimedia content types, including 2D videos, 3D videos, images, audio clips, songs, and music clips. The system can be deployed on private and/or public clouds. Our system has two novel components: (i) method to create signatures of 3D videos, and (ii) distributed matching engine for multimedia objects. The signature method creates robust and representative signatures of 3D videos that capture the depth signals in these videos and it is computationally efficient to compute and compare as well as it requires small storage. The distributed matching engine achieves high scalability and it is designed to support different multimedia objects. We implemented the proposed system and deployed it on two clouds: Amazon cloud and our private cloud. Our experiments with more than 11,000 3D videos and 1 million images show the high accuracy and scalability of the proposed system. In addition, we compared our system to the protection system used by YouTube and our results show that the YouTube protection system fails to detect most copies of 3D videos, while our system detects more than 98% of them. This comparison shows the need for the proposed 3D signature method, since the state-of-the-art commercial system was not able to handle 3D videos.

Index Terms

M. Hefeeda and T. ElGamal are with Qatar Computing Research Institute, Qatar (QCRI).

K. Calagari and A. Abdelsadek are with the School of Computing Science, Simon Fraser University, Canada. They were visiting QCRI as interns during this work.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TMM.2015.2389628, IEEE Transactions on Multimedia

Video copy detection, cloud applications, 3D video, video fingerprinting, depth signatures

I. Introduction

Advances in processing and recording equipment of multimedia content as well as the availability of free online hosting sites have made it relatively easy to duplicate copyrighted materials such as videos, images, and music clips. Illegally redistributing multimedia content over the Internet can result in significant loss of revenues for content creators. Finding illegally-made copies over the Internet is a complex and computationally expensive operation, because of the sheer volume of the available multimedia content over the Internet and the complexity of comparing content to identify copies.

We present a novel system for multimedia content protection on cloud infrastructures. The system can be used to protect various multimedia content types, including regular 2D videos, new 3D videos, images, audio clips, songs, and music clips. The system can run on private clouds, public clouds, or any combination of public-private clouds. Our design achieves rapid deployment of content protection systems, because it is based on cloud infrastructures that can quickly provide computing hardware and software resources. The design is cost effective because it uses the computing resources on demand. The design can be scaled up and down to support varying amounts of multimedia content being protected.

The proposed system is fairly complex with multiple components, including: (i) Crawler to download thousands of multimedia objects from online hosting sites, (ii) Signature method to create representative fingerprints from multimedia objects, and (iii) distributed matching engine to store signatures of original objects and match them against query objects. We propose novel methods for the second and third components, and we utilize off-the-shelf tools for the crawler. We have developed a complete running system of all components and tested it with more than 11,000 3D videos and 1 million images. We deployed parts of the system on the Amazon cloud with varying number of machines (from 8 to 128), and the other parts of the system were deployed on our private cloud. This deployment model was used to show the flexibility of our system, which enables it to efficiently utilize varying computing resources and minimize the cost, since cloud providers offer different pricing models for computing and network resources. Through extensive experiments with real deployment, we show the high accuracy (in terms of precision and recall) as well as the scalability and elasticity of the proposed system.

The contributions of this paper are as follows:

- Complete multi-cloud system for multimedia content protection. The system supports different types of multimedia content and can effectively utilize varying computing resources.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TMM.2015.2389628, IEEE Transactions on Multimedia

- Novel method for creating signatures for 3D videos. This method creates signatures that capture the depth in stereo content without computing the depth signal itself, which is a computationally expensive process.
- New design for a distributed matching engine for high-dimensional multimedia objects. This design provides the primitive function of finding K-nearest neighbors for large-scale datasets. The design also offers an auxiliary function for further processing of the K neighbors. This two-level design enables the proposed system to easily support different types of multimedia content. For example, in finding video copies, the temporal aspects need to be considered in addition to matching individual frames. This is unlike finding image copies. Our design of the matching engine employs the MapReduce programming model.
- Rigorous evaluation study using real implementation to assess the performance of the proposed system and compare it against the closest works in academia and industry. Specifically, we evaluate the entire end-to-end system with 11,000 3D videos downloaded from YouTube. Our results show that a high precision, close to 100%, with a recall of more than 80% can be achieved even if the videos are subjected to various transformations such as blurring, cropping, and text insertion. In addition, we compare our system versus the Content ID system used by YouTube to protect videos. Our results show that although the Content ID system provides robust detection of 2D video copies, it fails to detect copies of 3D videos when videos are subjected to even simple transformations such as re-encoding and resolution change. Our system, on the other hand, can detect almost all copies of 3D videos even if they are subjected to complex transformations such as synthesizing new virtual views and converting videos to anaglyph and 2D-plus-depth formats.

  Furthermore, we isolate and evaluate individual components of our system. The evaluation of the new 3D signature method shows that it can achieve more than 95% precision and recall for stereoscopic content subjected to 15 different video transformations; several of them are specific to 3D videos such as view synthesis. The evaluation of the distributed matching engine was done on the Amazon cloud with up to 128 machines. The engine was used to manage up to 160 million data points, each with 128 dimensions, extracted from over 1 million images. The results show that our design of the matching engine is elastic and scalable. They also show that our system outperforms the closest object matching system in the literature, called RankReduce, by a wide margin in accuracy and it is more efficient in terms of space and computation.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TMM.2015.2389628, IEEE Transactions on Multimedia

The rest of this paper is organized as follows. We summarize the related works in Section II. In Section III, we present the design goals and a high-level description of the proposed system. In Section IV, we present the details of the proposed signature creation method. In Section V, we describe the design of the matching engine. Our implementation and rigorous evaluation of the whole system as well as its individual components are presented in Section VI. We conclude the paper in Section VII.

II. Related Work

The problem of protecting various types of multimedia content has attracted significant attention from academia and industry. One approach to this problem is using watermarking [11], in which some distinctive information is embedded in the content itself and a method is used to search for this information in order to verify the authenticity of the content. Watermarking requires inserting watermarks in the multimedia objects *before* releasing them as well as mechanisms/systems to find objects and verify the existence of correct watermarks in them. Thus, this approach may not be suitable for already-released content without watermarks in them. The watermarking approach is more suitable for the somewhat controlled environments, such as distribution of multimedia content on DVDs or using special sites and custom players. Watermarking may not be effective for the rapidly increasing online videos, especially those uploaded to sites such as YouTube and played back by any video player. Watermarking is not the focus of this paper.

The focus of this paper is on the other approach for protecting multimedia content, which is content-based copy detection (CBCD) [16]. In this approach, signatures (or fingerprints) are extracted from original objects. Signatures are also created from query (suspected) objects downloaded from online sites. Then, the similarity is computed between original and suspected objects to find potential copies. Many previous works proposed different methods for creating and matching signatures. These methods can be classified into four categories: spatial, temporal, color and transform-domain. Spatial signatures (particularly the block-based) are the most widely used. However, their weakness is the lack of resilience against large geometric transformations. Temporal and color signatures are less robust and can be used to enhance spatial signatures. Transform-domain signatures are computationally intensive and not widely used in practice. For more details, see surveys for audio fingerprinting [5] and 2D video fingerprinting [16].

Youtube Content ID [10], Vobile VDNA [25] and MarkMonitor [18] are some of the industrial examples which use fingerprinting for media protection, while methods such as [13] can be referred to as the academic state-of-the-art. Unlike previous works, the contribution of this paper is to design a large-scale

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TMM.2015.2389628, IEEE Transactions on Multimedia

system to find copies that can be used for different types of multimedia content and can leverage multi-cloud infrastructures to minimize the cost, expedite deployment, and dynamically scale up and down. That is, we design our system such that previous content-based copy detection methods for creating and matching signatures can be implemented within our system.

In addition to our cloud-based system, we propose a new method for 3D video fingerprinting, and a new design for the distributed matching engine. The works related to each of these components are summarized in the following subsections.

A. 3D Video Signatures

Content-based copy detection of 3D videos is a fairly new problem; we are aware of only two previous works [21] and [12]. The work in [21] computes SIFT points in each view and uses the number of matching SIFT points to verify matches. Comparing all SIFT points in each frame is not practical for large databases due to the storage overhead and search complexity. On the other hand, the work in [12] assumes that the depth maps are given or estimated. Estimating the depth map from stereoscopic videos is quite expensive. The method in [12] is suitable for 3D videos encoded in the video plus depth format, but not for stereoscopic videos. Our proposed method in this paper captures the depth properties without calculating the depth map itself and it is computationally efficient because it does not compare all features in the frame.

Although 3D copy detection methods are scarce in the literature, there are many methods available for 2D video copy detection. Hampapur et al. [9] use the temporal features of the video as the signature. Similarly, Tasdemir et al. [23] use motion vectors as the signature for each frame. Some methods use color histograms as signatures, e.g., [9]. The color histogram signature is prone to global variations in color which are common when recoding video. Another group of methods use interest points of video frames as signature. For example, Liu et al. [15] use local SIFT features as the frame signature. Using gradient information has also shown to be robust to many 2D transformations [13].

All of the above 2D video fingerprinting methods can be implemented in the proposed system. In addition, while some of these methods can be used for 3D video copy detection, they are designed for 2D videos, and they ignore the information in different views and the depth of 3D videos. This information is important especially in the presence of 3D video transformations such as view synthesis, where views from different viewpoints can be generated using the depth map of the 3D video. When two new views are synthesized, the positioning of each pixel in the frame is changed, and some areas are occluded while other areas become visible. The luminance, gradient, color and even the interest points in each block can change as well when a new view is synthesized. Thus, the extracted signature using any of the 2D methods will change accordingly. Therefore, when searching for similar signatures, manipulated versions may not be identified. The importance of using signatures that have some information from the depth signal has been shown in [12]. In addition, our experiments and comparisons in this paper show that the state-of-the-art copy detection system used by YouTube (called Content ID) fails to detect many simple transformations made on 3D videos such as re-encoding, conversion to row or column interleaved formats, and creating new virtual views. Based on the available information from the patent describing the Content ID system [10] and our own experiments, we believe that the poor performance of Content ID on 3D videos is because it does not consider any depth information.

B. Distributed Matching Engine

Unlike many of the previous works, e.g., [3] which designed a system for image matching, our proposed matching engine is general and it can support different types of multimedia objects, including images, 2D videos, and 3D videos. To achieve this generality, we divide the engine into two main stages. The first stage computes nearest neighbors for a given data point, and the second stage post-processes the computed neighbors based on the object type. In addition, our design supports high-dimensionality which is needed for multimedia objects that are rich in features.

Computing nearest neighbors is a common problem in many applications. Our focus in this paper is on distributed techniques that can scale to large datasets such as [14], [17], [3], [22]. Liao et al. [14] build a multi-dimensional index using R-tree on top of the Hadoop distributed file system (HDFS). Their index, however, can only handle low dimensional datasets—they performed their experiments with two dimensional data. They solve the K nearest neighbors over large datasets using MapReduce [7]. Lu et al. [17] construct a Voronoi-like diagram using some selected pivot objects. They then group the data points around the closest pivots and assign them to partitions, where searching can be done in parallel. The system in [17] is also designed for low dimensional datasets; it did not consider data with more than 30 dimensions. In contrast, in our experiments we used images and videos with up to 128 dimensions. Aly et al. [3] propose a distributed system for image retrieval. A major drawback of this system is using a single root machine that directs all query points, which makes it a single point of failure as well as a bottleneck that could slow down the whole system. Our system does not use a central node, and thus it is more robust and scalable.

The closest work to ours is the RankReduce system [22], which implements a distributed LSH (Locality Sensitive Hashing) index on a computing cluster using MapReduce. RankReduce maintains multiple hash

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TMM.2015.2389628, IEEE Transactions on Multimedia

tables over a distributed cluster, which requires storing multiple replicas of the datasets in hash tables. This incurs significant storage cost and it increases the number of I/O operations. In contrast, our system stores the dataset only once. We compare the proposed matching engine against RankReduce and we show that our system returns more accurate neighbors and it is more efficient.

III. Overview of the Proposed System

The goal of the proposed system for multimedia content protection is to find illegally made copies of multimedia objects over the Internet. In general, systems for multimedia content protection are large-scale and complex with multiple involved parties. In this section, we start by identifying the design goals for such systems and our approaches to achieve them. Then, we present the high-level architecture and operation of our proposed system.

A. Design Goals and Approaches

A content protection system has three main parties: (i) content owners (e.g., Disney), (ii) hosting sites (e.g., YouTube), and (iii) service providers (e.g., Audible Magic). The first party is interested in protecting the copyright of some of its multimedia objects, by finding whether these objects or parts of them are posted on hosting sites (the second party). The third party is the entity that offers the copy finding service to content owners by checking hosting sites. In some cases the hosting sites offer the copy finding service to content owners. An example of this case is YouTube, which offers content protection services. And in other, less common, cases the content owners develop and operate their own protection systems.

We define and justify the following four goals as the most important ones in multimedia content protection systems.

- *Accuracy:* The system should have high accuracy in terms of finding all copies (high recall) while not reporting false copies (high precision). Achieving high accuracy is challenging, because copied multimedia objects typically undergo various modifications (or transformations). For example, copied videos can be subjected to cropping, embedding in other videos, changing bit rates, scaling, blurring, and/or changing frame rates. Our approach to achieve this goal is to extract signatures from multimedia objects that are *robust* to as many transformations as possible.
- *Computational Efficiency:* The system should have short response time to report copies, especially for timely multimedia objects such as sports videos. In addition, since many multimedia objects are continually added to online hosting sites, which need to be checked against reference objects, the content protection system should be able to process many objects over a short period of time. Our

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TMM.2015.2389628, IEEE Transactions on Multimedia

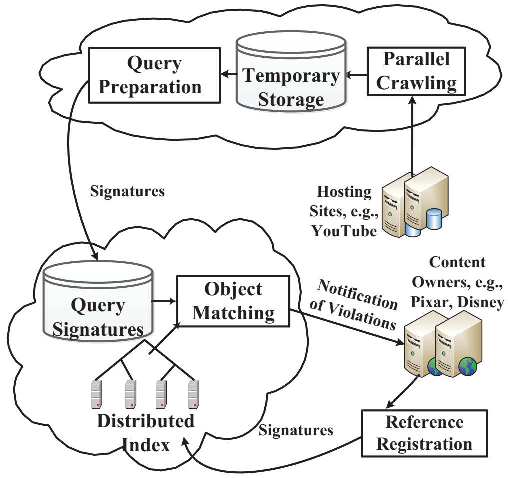


Fig. 1. Proposed cloud-based multimedia content protection system.

approach to achieve this goal is to make the signatures compact and fast to compute and compare without sacrificing their robustness against transformations.

- *Scalability and Reliability:* The system should scale (up and down) to different number of multimedia objects. Scaling up means adding more objects because of monitoring more online hosting sites, having more content owners using the system, and/or the occurrence of special events such as sports tournament and release of new movies. Conversely, it is also possible that the set of objects handled by the system shrinks, because, for example, some content owners may terminate their contracts for the protection service. Our approach to handle scalability is to design a distributed system that can utilize varying amounts of computing resources.

  With large-scale distributed systems, failures frequently occur, which require the content protection system to be reliable in face of different failures. To address this reliability, we design the core parts of our system on top of the MapReduce programming framework, which offers resiliency against different types of failures.
- *Cost Efficiency:* The system should minimize the cost of the needed computing infrastructure. Our approach to achieve this goal is to design our system to effectively utilize cloud computing infrastructures (public and/or private). Building on a cloud computing infrastructure also achieves the scalability objective discussed above and reduces the upfront cost of the computing infrastructure.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TMM.2015.2389628, IEEE Transactions on Multimedia

B. Architecture and Operation

The proposed cloud-based multimedia content protection system is shown in Figure 1. The system has multiple components; most of them are hosted on cloud infrastructures. The figure shows the general case where one or more cloud providers can be used by the system. This is because some cloud providers are more efficient and/or provide more cost saving for different computing and communication tasks. For example, a cloud provider offering lower cost for inbound bandwidth and storage can be used for downloading and temporarily storing videos from online sites (top cloud in the figure), while another cloud provider (or private cloud) offering better compute nodes at lower costs can be used to maintain the distributed index and to perform the copy detection process (lower cloud in the figure).

The proposed system can be deployed and managed by any of the three parties mentioned in the previous section: content owners, hosting sites, or service providers. The proposed system has the following main components, as shown in Figure 1:

- Distributed Index: Maintains signatures of objects that need to be protected.
- Reference Registration: Creates signatures from objects that content owners are interested in protecting, and inserts them in the distributed index.
- Query Preparation: Creates signatures from objects downloaded from online sites, which are called query signatures. It then uploads these signatures to a common storage.
- Object Matching: Compares query signatures versus reference signatures in the distributed index to find potential copies. It also sends notifications to content owners if copies are found.
- Parallel Crawling: Downloads multimedia objects from various online hosting sites.

The Distributed Index and Object Matching components form what we call the Matching Engine, which is described in Section V. The second and third components deal with signature creation, which is described in Section IV. For the Crawling component, we designed and implemented a parallel crawler and used it to download videos from YouTube. The details of the crawler are omitted due to space limitations.

The proposed system functions as follows. Content owners specify multimedia objects that they are interested in protecting. Then, the system creates signatures of these multimedia objects (called reference objects) and inserts (registers) them in the distributed index. This can be one time process, or a continuous process where new objects are periodically added. The Crawl component periodically (e.g., once a day) downloads recent objects (called query objects) from online hosting sites. It can use some filtering (e.g., YouTube filtering) to reduce the number of downloaded objects. For example, for video objects, it can

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TMM.2015.2389628, IEEE Transactions on Multimedia

download videos that have a minimum number of views or belong to specific genre (e.g., sports). The signatures for a query object are created once the Crawl component finishes downloading that object and the object itself is removed. After the Crawl component downloads all objects and the signatures are created, the signatures are uploaded to the matching engine to perform the comparison. Compression of signatures can be performed before the upload to save bandwidth. Once all signatures are uploaded to the matching engine, a distributed operation is performed to compare all query signatures versus the reference signatures in the distributed index.

IV. Signature Creation

The proposed system is designed to handle different types of multimedia objects. The system abstracts the details of different media objects into multi-dimensional signatures. The signature creation and comparison component is media specific, while other parts of the system do not depend on the media type. Our proposed design supports creating composite signatures that consist of one or more of the following elements:

- Visual signature: created based on the visual parts in multimedia objects and how they change with time.
- Audio signature: created based on the audio signals in multimedia objects.
- Depth signature: if multimedia objects are 3D, signatures from their depth signals are created.
- Meta data: created from information associated with multimedia objects such as their names, tags, descriptions, format types, and IP addresses of their uploaders or downloaders.

Previous works have addressed creating visual signatures for 2D videos [16] and audio signals [5]. These works and others can be supported by our system in a straightforward manner. In the current paper, we present a novel method for creating depth signatures from *stereoscopic* 3D videos, which are the most common format of 3D videos nowadays. In such videos, the depth signal is not explicitly given. Rather, the video is presented in two views. Our method computes a signature of the depth signal *without* computing the depth signal itself.

The proposed method takes as input a 3D video encoded in stereo format, which is composed of two views (left view for left eye and right view for right eye). Each view is a stream of frames which correspond to frames in the other view. The output of the method is a signature for each pair of frames. To reduce the computation and storage costs, subsampling can be applied in which signatures are computed only for a subset of frames, e.g., every tenth frame.

The proposed method is composed of the following main steps.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TMM.2015.2389628, IEEE Transactions on Multimedia

- *Step 1: Compute Visual Descriptors for Left and Right Images.* Visual descriptors are local features that describe salient parts of an image. Different types of descriptors can be used, including, SURF, SIFT, and HOG (Histogram of Oriented gradients). The default descriptor used in our method is SURF. Each descriptor has a fixed number of dimensions or features. For example, each SURF descriptor has 64 dimensions. Each descriptor i is computed at a specific pixel in the image, which has a location of (x_i, y_i). The result of this step is two sets of descriptors; one for the left image and one for the right image:

$$D_i^L = (f_{i1}, f_{i2}, \dots, f_{iF}), i = 1, 2, \dots, L_n, \tag{1}$$

$$D_j^R = (f_{j1}, f_{j2}, \dots, f_{jF}), j = 1, 2, \dots, R_n, \tag{2}$$

  where L_n and R_n are the number of descriptors in the left and right images, respectively and F is the number of dimensions in each descriptor.
- *Step 2: Divide each Image into Blocks.* Both the left and right images are divided into the same number of blocks. In general, blocks can be of different sizes and each can be a square or other geometrical shape. In our implementation, we use equal-size square blocks. Thus, each image is divided into $N \times M$ blocks.
- *Step 3: Match Visual Descriptors.* For each visual descriptor in the left image, we find the closest descriptor in the right image. We consider the block that the descriptor is located in and we find its corresponding block in the right image. We draw a larger window around this corresponding block. This is done to account for any slight changes in the visual objects between the left and right views. Different types of similarity measures can be used to compute the distance between feature vectors. In our implementation, we use the Euclidean distance to compute the distance between descriptors:

$$D_i^L - D_j^R = \sqrt{(f_{i1} - f_{j1})^2 + \cdots + (f_{iF} - f_{jF})^2}. \tag{3}$$

  We compute the distance between each visual descriptor in the left image and all descriptors in the corresponding block of the right image. The corresponding match is the descriptor with the smallest distance.
- *Step 4: Compute Block Disparity.* We compute the block disparity between each block in the left image and its corresponding block in the right image. The disparity of a single descriptor i is given by:

$$\sqrt{((x_i - x_j)/W_b)^2 + ((y_i - y_j)/H_b)^2}, \tag{4}$$

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TMM.2015.2389628, IEEE Transactions on Multimedia

where (x_i, y_i) is the position of descriptor i in the left image, and (x_j, y_j) is the position of the corresponding descriptor j in the right image. We normalize the disparity by the width W_b and the height H_b of each block in the image. The disparity of block b_i is denoted by S_{b_i} and computed as the average disparity of all visual descriptors in that block. If a block or its corresponding block in the right image does not have any descriptor, the disparity is set to 0.

- *Step 5: Compute Signature.* The signature of the two corresponding images is: $(S_{b_1}, S_{b_2}, \ldots, S_{b_{N \times M}})$. We note that the signature is compact and fixed in size as the total number of blocks $N \times M$ is fixed and small (in our experiments, we have 5×5 blocks).

In summary, our method constructs *coarse-grained* disparity maps using stereo correspondence for a sparse set of points in the image. Stereo correspondence tries to identify a part in an image that corresponds to a part in the other image. A fine-grained disparity map of a pair of images describes the displacement needed for each pixel to move from one image to the correct position in the other image. The disparity map is inversely proportional to the depth map, meaning that the disparity is larger for objects near the camera than objects far away from the camera. Since fine-grained disparity maps are expensive to compute, we create our signature from coarse-grained disparity maps, which are computed from blocks of pixels.

V. Distributed Matching Engine

We design a matching engine suitable for different types of multimedia objects that is scalable and elastic. Scalability is needed to handle large datasets with millions of multimedia objects. Elasticity is a desirable feature that allows our system to utilize varying amount of computing resources offered on cloud infrastructures.

In general, multimedia objects are characterized by many features and each feature is of high dimensions. For example, an image can be characterized by 100–200 SIFT descriptors, and each has up to 128 dimensions, and a video object will have even more features extracted from its frames. In addition, different types of multimedia objects require different number of features as well as different processing operations in order to decide on object matching. For example, matching two video clips requires not only matching individual frames, but also the temporal sequence of these frames. This is unlike image matching. To address this generality, we design the matching engine as two stages. In the first stage, the engine provides an efficient, distributed, implementation for computing K nearest neighbors for high-dimensional data. In the second stage, the engine provides a generic interface for post processing these neighbors based on the different needs of various media types and applications. For instance, for video

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TMM.2015.2389628, IEEE Transactions on Multimedia

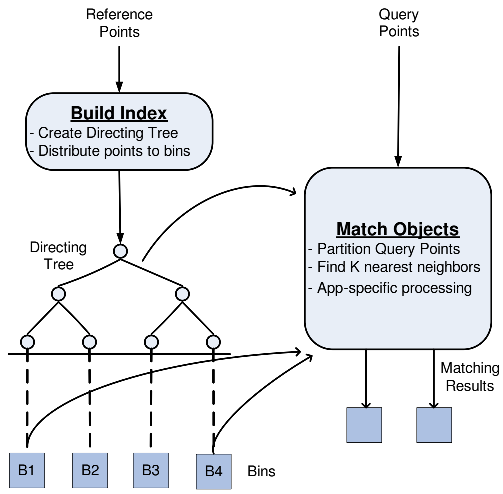


Fig. 2. High-level architecture of the distributed index component used in the multimedia content protection system. Round boxes are MapReduce jobs.

copy protection, the individual frame matching is done in the first stage and the temporal aspects are considered in the second stage. For image protection, the second stage can be empty.

The matching engine is implemented using the MapReduce distributed programming model [7]. The design is not restricted to MapReduce and can be implemented in other distributed programming platforms. MapReduce provides an infrastructure that runs on a cluster of machines, which automatically manages the execution of multiple computations in parallel as well as the communications among these computations. This distributed design allows the index to scale to large volumes of data and to use variable amounts of computational resources. It also provides transparent redundancy and fault tolerance to computations. The following subsections briefly describe the index construction and object matching operations. More details can be found in [1] and our preliminary work in [2].

A. Constructing the Matching Engine

The basic design of the matching engine is illustrated in Figure 2. It has a data structure that we call the distributed index as well as distributed processing operations. The index is divided into two parts: (i) Directing tree and (ii) Bins. Directing tree is a space partitioning tree [20] that is used to group similar points in the same or close-by bins. It is also used to forward query points to bins with potential matches. Bins are leaf nodes of the directing tree, but they are stored as files on the distributed file system. All processing of the matching engine is performed in two distributed operations: (i) Build Index, and (ii)

Match Objects. The first creates the index from reference data points, and the second matches query objects versus reference objects in the index.

The design of our index has two main features that make it simple to implement in a distributed manner, yet efficient and scalable. First, data points are stored only at leaf nodes. Intermediate nodes do not store any data, they only store meta data to guide the search through the tree. This significantly reduces the size of the directing tree and makes it fit easily in the main memory of a single machine even for large datasets. This feature allows us to distribute copies of the directing tree to distributed machines to process queries in parallel. Replicating the directing tree on different machines not only facilitates distributed processing, but it also greatly improves the robustness and efficiency of the system. The robustness is improved because there is no single point of failures. The efficiency is improved because there is no central machine or set of machines that other machines need to contact during the computation. The second feature of our index design is the separation of leaf nodes (bins) and storing them as files on the distributed file system. This increases reliability as well as simplifies the implementation of the distributed computations in our system, because concurrent accesses of data points are facilitated by the distributed file system.

The distributed index is constructed from reference objects, which is done before processing any queries. Constructing the index involves two steps: (i) creating the directing tree and (ii) distributing the reference dataset to bins. Once created, the directing tree is serialized as one object and stored on the distributed file system. This serialized object can be loaded in memory by various computational tasks running on multiple machines in parallel. Distribution of data is done in parallel on multiple machines using a simple MapReduce job.

The directing tree is the top part of the index, which contains all non-leaf nodes. Different types of trees [20] can be used as directing tree, after we perform our ideas of keeping data points only at leaves, aggregating data points into bins, and storing bins on the distributed file system. We chose the KD tree [4] as the base for our directing tree, because of its efficiency and simplicity. A KD tree is a binary tree in which every node is a K-dimensional point. Every non-leaf node can be considered as a splitting hyperplane that divides the space into two parts. Points to the left of this hyperplane represent the left sub-tree of that node and points to the right of the hyperplane represent the right sub-tree. The hyperplane direction is chosen in a way such that every node in the tree is associated with one of the K dimensions, with the hyperplane perpendicular to that dimension's axis, and it splits the data points around it into two equal-size subsets. The equal-size subsets make the tree balanced.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TMM.2015.2389628, IEEE Transactions on Multimedia

We are interested in matching objects with high dimensions. Thus, if we use the traditional KD tree, it will be too deep with too many leaf nodes and each has only one data point, which is not efficient especially in distributed processing environment where accessing any node may involve communications over the network. We control the depth of the tree based on the size of the dataset such that the size of bins at the bottom of the tree roughly matches the storage block size of the distributed file system. In real deployment, the size of a leaf node is in the order of 64 to 128 MBs, which means that each leaf node will contain thousands of data points. Thus, the size of our directing tree will be small. Since we compress the depth of the tree, we use only a subset of the dimensions of the data points. We use the principal component analysis (PCA) to choose the most representative dimensions to project the dataset on. PCA is a well studied technique for dimension reduction. It finds a hyperplane of the required target dimensionality to project the actual points on, such that the variance among them after projection is maximized. It finds this hyperplane by calculating the singular value decomposition (SVD) of the covariance matrix of the input points.

B. Matching Objects

The object matching process is done in three steps: (i) partitioning query dataset, (ii) finding K nearest neighbors for each data point in the query dataset, and (iii) performing application-specific object matching using the found K nearest neighbors. Each of these three steps is executed in parallel on the MapReduce infrastructure. The first step partitions the query dataset such that each partition contains a bin and a list of data points that are likely to have neighbors in that bin. This is done using the directing tree, which is used to create the list of data points that corresponds to each bin.

The second and third steps of the object matching process first find the K nearest neighbors and then apply application-specific function(s) on them to produce the final object matching results. These steps are achieved through one MapReduce job that has one mapper and two consecutive reducers. The mapper and first reducer compute the K nearest neighbors for all points in the query dataset. The second reducer performs various post processing functions on the K nearest neighbors based on the multimedia object type.

VI. Evaluation

We have implemented and integrated all parts of the proposed content protection system: from a web user interface to control various parts of the system and its configurations, to tools to allocate, release, and manage cloud resources, to all algorithms for creating and matching signatures, as well as all distributed

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TMM.2015.2389628, IEEE Transactions on Multimedia

MapReduce algorithms for processing thousands of multimedia objects. This is a fairly complex system with tens of thousands of lines of code in different programming and scripting languages.

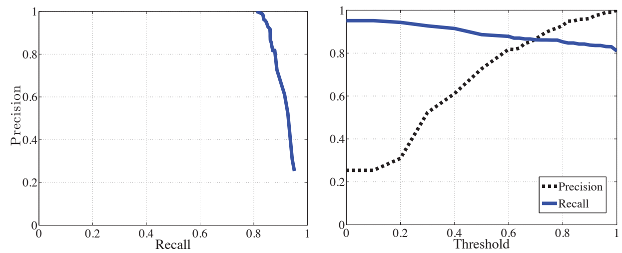
We validated our proposed multi-cloud architecture by deploying part of our system on the Amazon cloud and the other part on our local private cloud. The Amazon cloud had up to 20 machines and our private cloud had 10 machines each with 16 cores. We deployed the Parallel Crawling and Query Preparation components on the Amazon cloud. This is because the Amazon cloud has large Internet links and it can support downloading thousands of multimedia objects from various sites, such as YouTube. The relatively close proximity and good connectivity of Amazon data centers in North America to major multimedia content hosting sites accelerates the download process. More importantly, at the time of our experiments, the Amazon pricing model did not charge customers for inbound bandwidth while it charged for outbound bandwidth. Since the majority of our workload is downloading multimedia objects (inbound traffic), this deployment minimized our costs, and it indeed shows the benefits of our architecture, which can opportunistically utilize resources from different clouds. After downloading each multimedia object, we create signatures from it and immediately delete the object itself as it is no longer needed–we keep the object URL link on the hosting site from which we downloaded it. This minimizes our storage cost on Amazon. Signatures from multiple multimedia objects are then grouped, compressed, and transferred to our private cloud for more intensive processing. Once uploaded to the private cloud, the signatures are deleted from Amazon to save storage. On our private cloud, we deploy the matching engine and all of its related operations. These include building the distributed index from reference objects and matching query objects versus reference objects in the index. The crawling and matching operations are done periodically; in our system we do it once daily, when our local cloud is lightly loaded.

We rigorously evaluate the proposed system using real deployment with thousands of multimedia objects. Specifically, in the following subsections, we evaluate our system from four angles: (i) complete system performance, (ii) comparison with YouTube, (iii) analysis of the signature method, and (iv) accuracy, scalability and elasticity of the distributed matching engine component.

A. *Performance of the Complete System*

Videos. We assess the performance of the whole system with a large dataset of 11,000 3D videos downloaded from YouTube. These videos are from different categories, and have diverse sizes, durations, resolutions, and frame rates. Thus, they represent a good sample of most 3D videos on YouTube. 10,000 of these videos make the bulk of our query set, while the other 1,000 videos make the bulk of our reference set. We downloaded both the 10,000 query videos and the 1,000 reference videos in a similar

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TMM.2015.2389628, IEEE Transactions on Multimedia



(a) Precision–Recall curve (b) Precision and recall vs. threshold s

Fig. 3. Performance of the complete system for 3D video copy protection on more than 11,000 3D videos.

manner as follows, while keeping a list of all previously downloaded video IDs to ensure that the reference set does not include any of the downloaded query videos. First, we used the APIs provided by YouTube to download the top 100 videos in terms of view count in each video category. YouTube has 22 categories: Music, Entertainment, Sports, Film, Animation, News, Politics, Comedy, People, Blogs, Science, Technology, Gaming, Howto, Style, Education, Pets, Animals, Autos, Vehicle, Travel, and Events. Since some categories did not have any 3D videos and some of them had a small number, we added a set of seed queries to expand our dataset. The queries we used were: 3d side by side, 3d trailer, 3d landscape, 3d animation, 3d football, 3d gaming and 3d ads.

For the reference video set, in addition to the 1,000 downloaded videos, we used 14 other videos that were *manually* downloaded to be as diverse as possible and the likelihood of them being in the query set is low. We chose 10 out of these 14 videos and manipulated each of them using the `ffmpeg` video processing tool in five different ways: cutting clips or segments, scaling, blurring, logo insertion, and cropping. Thus, we created 50 videos in total. We added these 50 manipulated videos to the query set to ensure that the query set has matches to some of the videos in the reference set, which made the query set have 10,050 videos. We created signatures from all reference videos and inserted them in the matching engine.

Methodology. For each frame of the query video, the signature is computed and the closest signatures to it are retrieved from the distributed index. Candidate matches undergo an additional step to ensure that the number of matching frames in the two videos is enough. For example, if frame x in the query video matches frame y in the reference video, we expect frame $x+1$ in the query video to match frame $y+1$ in the reference video. In order to consider this, a matching matrix is computed for each pair of candidate reference video and query video. The size of a matching matrix is the number of frames in the considered

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TMM.2015.2389628, IEEE Transactions on Multimedia

reference video times the number of frames in the query video against which the reference video is being compared. A value of 1 in the (i, j) position of the matching matrix means that the i^{th} frame of the reference video has matched the j^{th} frame of the query video. The longest diagonal sequence of 1s in this matrix indicates the largest number of matching frames and is considered as a potential copy. The final matching score between videos is the number of matches on the diagonal divided by the diagonal length. We introduce a threshold parameter s to decide whether two videos match. If two videos have a matching score less than s, they are not considered a match; otherwise they are a match. We vary the threshold s between 0 and 1.0.

We measure the performance in terms of two basic metrics: precision (percentage of returned videos that are true copies) and recall (percentage of true video copies that are returned). In this large experiment, we compute the exact precision, which is possible as we can check whether a match declared by the system is a true match or not by watching the videos. Computing the exact recall is tricky though, since we cannot be 100% sure that the large query set does not contain any copies other than the added 50 videos, although we tried to minimize this possibility. The only way to be sure is to manually check each of the 10,000 videos, which is a formidable task. To *partially* mitigate this issue, we compute an *approximation* of the recall assuming that there are no other copies in the 10,000 videos. Thus, the computed recall should be viewed as an *upper bound* on the achievable recall of our system. We compute the exact recall on small datasets in later sections.

Results. We plot the results of this experiment in Figure 3. Figure 3(a) shows the precision-recall (PR) curve, where we plot the approximate recall as discussed above. To get this PR curve, we change the threshold from 0 to 1, and compute the precision and recall for each threshold. PR curves are a standard evaluation method in image retrieval, as they contain rich information and can easily be read by researchers [19]. The results clearly show that our system can achieve both high precision and recall. For example, a precision of 100% with a recall of more than 80% can be achieved. To further analyze the results, we show in Figure 3(b), how the precision and recall vary with the threshold parameter s. The results show that our method can achieve precision and recall values of more than 80% for a wide range of thresholds from 0.6 to 1. This means that our system does not only provide high accuracy, but it is not very sensitive to the threshold s, which is an internal system parameter. In other words, the system administrator does not need to accurately fine tune s.

In summary, this large-scale experiment with 11,000+ 3D videos and the whole system deployed on multiple distributed machines confirm the accuracy of the proposed system.

B. Comparison with YouTube

YouTube is one of the largest online video sharing and streaming sites in the world. It offers a video protection service to its customers, which employs a sophisticated system to detect illegal copies of protected videos. The system used in YouTube is called Content ID [6], which is a proprietary system and we cannot know much details about it beyond what YouTube disclosed in its patent [10]. The goal of this subsection is not to conduct full comparison between our system and Content ID, which is not possible. Our goal is to show that while the Content ID system provides robust copy detection for traditional 2D videos, it fails to detect most copies of 3D videos and that the proposed system, which employs our new 3D signature method, outperforms Content ID by a large margin.

Methodology. To test the Content ID system, we download several *copyrighted* 2D and 3D videos from YouTube. We perform various transformations on these videos and then upload them back to YouTube to see whether the Content ID system can detect them as copies. In case of detection, YouTube shows the message “Matched third party content” when a copyrighted video is uploaded. Similarly, we test our system by using the same 3D videos downloaded from YouTube and subjected to the same transformations.

In particular, we downloaded six 3D and six 2D protected videos from Warner Bros. and 3net YouTube channels. The video lengths are in the range of 30 seconds to 2 minutes. When we uploaded each of these 12 videos back to YouTube *without* any modifications, the Content ID system correctly identified them all as copies.

Results for 2D Videos. We tested YouTube for detecting modified 2D videos. We applied six transformations on each of the six 2D videos: blur, format change, frame dropping, re-encoding with different resolution (scale), cutting 30 second clip, and cutting 40 second clip. Then, we uploaded all 36 modified versions of the 2D videos to YouTube. We found that the Content ID system can detect all blur, format change, frame dropping, and re-encoding with different resolution transformations as copies, resulting in a recall of 100% for these transformations. For the clip transformation, only one of the 30 seconds clips was detected, but all 40 seconds clips were detected as copies. Therefore, this experiment shows that the YouTube Content ID is quite robust for 2D videos.

Results for 3D Videos. Now, we test the Content ID system on 3D videos and compare it against our system. Recall that our system is general and can support different types of media, but we focus in this paper on designing signatures for 3D videos as there have been little work on these videos, whereas there are many works for 2D copy detection. The original 3D videos downloaded from YouTube are in

TABLE I

COMPARISON AGAINST YOUTUBE IN TERMS OF RECALL.

Transformation	YouTube	Proposed System
Blur	0/6	6/6
File format change (mp4 to avi)	6/6	6/6
Re-encoding: same bit-rate	0/6	6/6
Re-encoding: different bit-rate	0/6	6/6
Re-encoding: different resolution	0/6	6/6
Frame dropping	0/6	6/6
30 seconds clip	1/6	6/6
35 seconds clip	2/6	6/6
40 seconds clip	4/6	6/6
45 seconds clip	5/6	6/6
Anaglyph	5/6	5/6
Row-interleaved	5/6	6/6
Column-interleaved	6/6	6/6
2D-plus-depth	0/6	6/6
View synthesis	0/6	6/6

side-by-side format. We applied 15 different transformations; the first ten of these transformations are common between 2D and 3D videos, while the other five are specific to 3D videos. The transformations on each 3D video are: blur, file format change, re-encoding with same bit-rate, re-encoding with different bit-rate, re-encoding with different resolution (scale), frame dropping, 30 seconds clip, 35 seconds clip, 40 seconds clip, 45 seconds clip, anaglyph, row-interleaved, column-interleaved, 2D-plus-depth, and view synthesis. Anaglyph means changing the video format such that the right and left images are encoded in different colors to render 3D perception on regular 2D displays using anaglyph glasses (color filters). Row and column-interleaved indicate changing the format of the left and right images to suit row- and column-interleaved types of displays. The 2D-plus-depth transformation computes a depth for the video and presents the video as 2D stream and depth stream, which can be rendered by certain types of 3D displays. View synthesis is used to create additional virtual views from the basic stereo video. This is done to enhance user's experience or to evade the copy detection process.

As in 2D videos, we uploaded all modified 90 (=15 x 6) 3D videos to YouTube in order to check whether the Content ID system can identify them as copies. We explicitly specified that these videos are 3D when we uploaded them to YouTube. The results are shown in the second column of Table I. The

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TMM.2015.2389628, IEEE Transactions on Multimedia

results clearly show the poor performance of the Content ID system on 3D videos. For example, the Content ID system could not detect even a single copy of the six 3D videos when they are subjected to seven different transformations. Some of these transformations are as simple as bluring and re-encoding while others are more sophisticated such as view synthesis and 2D-plus-depth conversion. Furthermore, except for few transformations, the Content ID system misses most of the modified copies. The three transformations anaglyph, row-interleaved, and column-interleave result in videos that are similar to their corresponding 2D versions. Since the 2D versions of the used 6 3D videos are also under copyright protection, the videos resulting from such transformations are most probably matched against the 2D versions of the original videos.

To assess the accuracy of our system, we use the same six 3D videos as our reference dataset. We also add the 14 videos mentioned in Section VI-A to the reference dataset. We apply the same 15 transformations on the six 3D videos, resulting in 90 query videos. We add 1,000 other 3D videos downloaded from YouTube to the query dataset. We add these *noise* videos in order to check whether our system returns false copies. We report the results from our system in column three of Table I. To be fairly comparable with Content ID, we only report the recall from our system that is achieved at 100% precision, since through all of our experiments with YouTube the precision was 100%. As Table I shows, our system was able to detect 89 out of the 90 modified copies of the 3D videos, including complex ones such as view synthesis.

In summary, the results in this section show that: (i) there is a need for designing robust signatures for 3D videos since the current system used by the leading company in the industry fails to detect most modified 3D copies, and (ii) our proposed 3D signature method can fill this gap, because it is robust to various transformations including new ones specific to 3D videos such as anaglyph and 2D-plus-depth format conversions as well as synthesizing new virtual views.

C. Performance of 3D Signature Creation Component

We conduct small-scale, controlled experiments to rigorously analyze the proposed 3D signature method. We need the experiment to be small because we manually modify videos in different ways and check them one by one. This is needed to compute the exact precision and recall of the proposed method.

The reference video set contains the 14 videos mentioned in Section VI-A. The query set is created by modifying some of the reference videos in many different ways. Specifically, we apply the following transformations:

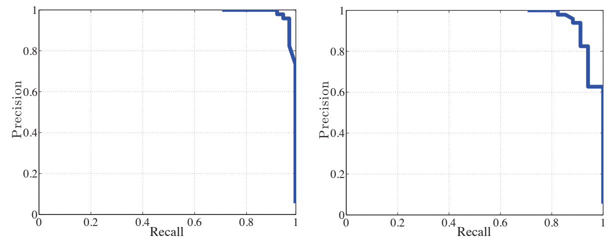
- Video Blurring: reduces the sharpness and contrast of the image. Radius of blur is in the range of [3,5].

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TMM.2015.2389628, IEEE Transactions on Multimedia

- Video Cropping: crops and discards part of an image. The discarded pixels are chosen at the boundaries. The number of discarded pixels is in the range [20,40].
- Video Scaling: reduces the resolution of the video and the scale factor is in the range [0.5,1.5].
- Logo Insertion: puts a logo on one of the corners of the video, the logo size is in the range of [20,40] pixels.
- Frame Dropping: periodically drops frames from the original video. The period is in the range [2,10], where 2 means every other frame is dropped and period 10 means every tenth frame is dropped. This transformation changes the video frame rate.
- Video Transcoding: changes the video from one file format to another.
- Text Insertion: writes random text on the video at different places.
- Anaglyph: multiplexes the left and right views of the 3D video over each other with different colors, typically red and blue.
- Row Interleaved: interleaves the left and right views of the 3D video horizontally row by row such that the odd rows belong to the left view and the even rows belong to the right view.
- Column Interleaved: interleaves the left and right views of the 3D video vertically column by column such that the odd columns belong to the left view and the even columns belong to the right view.
- 2D-plus-depth: converts the video from left and right views stacked together horizontally side-by-side into another format which is a 2D video and its associated depth.
- View Synthesis: uses the original left and right views to create another two virtual views to be used instead of the original views.

We conduct two types of experiments: (i) individual transformations, in which we study the effect of each video transformation separately, and (ii) multiple transformations, in which we assess the impact of multiple transformations applied to the same video. In the first individual transformations experiments, we apply the above listed individual transformations (except view synthesis) on each of the 14 videos mentioned in Section VI-A using `ffmpeg`. View synthesis is applied using the VSRS view synthesis tool [26]. View synthesis is applied on two videos other than the 14, which are Ballet and BreakDancers. We create 18 different versions from each of these two videos with synthesized views. In the multiple transformations experiments, we choose 10 videos and apply on each of them three transformations at the same time. These transformations are blurring, scaling and logo insertion. Although the combined transformations are not likely to occur in many videos, they show the robustness of our method. Applying

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TMM.2015.2389628, IEEE Transactions on Multimedia



(a) Single Transformation (b) Three Transformations

Fig. 4. The average accuracy of the proposed 3D signature method.

all these types of transformations on different videos results in a diverse and large query video set, which stresses our signature method.

Results for Individual Transformations. We first present the average results across all videos and all transformations. We plot the aggregate results in Figure 4(a) in the form of the precision versus recall curve. The precision-recall curve shows the achievable high accuracy of the proposed method. For example, a precision of 100% can be achieved with a recall up to 95%, by controlling the threshold value. To better understand the impact of the threshold parameter, we analyze the achieved average precision and recall by our method for all possible values of the threshold s. The figure is not shown due to space limitations. Our results show that the proposed method can concurrently achieve high precision and recall. For example, both the precision and recall are more than 90% for a large range of the threshold s. We note that this high accuracy is achieved when comparing *significantly modified* copies of videos versus reference videos. Our method achieves 100% accuracy (both precision and recall) when we compare unmodified versions of the videos against reference videos.

Next, we analyze the accuracy of the proposed signature method for each video transformation separately. This is to understand the robustness of the method against each transformation. We computed the precision-recall curve for each case. In addition, we computed the precision and recall for each value of the threshold. Due to space limitations, we omit these figures. The results show that our method is highly robust against the quite common logo insertion transformation as it can achieve 100% precision and recall at wide range of thresholds. This is because a logo can affect one or a few blocks in the video frames, which is a relatively small part of the signature. Similar high accuracy results are achieved for two other common transformations: video blurring and transcoding or format/bitrate change. In addition, for scaling, cropping, and frame dropping, our method still achieves fairly high accuracy. For example, our method is robust against video scaling. This is because during the creation of the signature, it is normalized by

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TMM.2015.2389628, IEEE Transactions on Multimedia

the frame resolution as described in Section IV. Moreover, for frame dropping our method achieves high precision and recall at low thresholds, which means that true matches are found but with low matching score, due to the gaps in the matching diagonal between both videos. Finally, for cropping, the results show that our signature method can identify 80% of the matches with 100% precision, or identify all matches with about 90% precision, which indicates that our method is also robust against cropping. This is because our 3D signature contains the depth values of each block. Since depth is usually smooth, neighboring blocks usually have similar values, causing our 3D signature to be less sensitive to block misalignments.

Results for Multiple Transformations. The aggregate results in the form of precision-recall curve are shown in Figure 4(b). The results clearly show the high accuracy of the proposed method, as high precision of more than 90% can be achieved with at least a recall of 90%. We note that the recall in this experiment is slightly less than the achieved recall in the previous experiment, because of the combined transformations applied in this case.

Running Time for 3D Signature Creation. In this experiment, we measure the running time of our 3D signature creation and compare it to the method in [12], which requires computing the depth maps. Thus, we chose a practical depth estimation method [24]. We compute the running time for only the depth estimation step of the signature creation method in [12] and compare it to the whole running time of our method. We run the experiment for 362 frames on the same machine. The results show that the average running time for our 3D signature creation is 0.87 sec, with minimum and maximum values of 0.39 sec and 1.62 sec, respectively. Whereas the average running time of depth estimation step alone is 68.91 sec, ranging from 61.26 sec and up to 85.59 sec.

It can be seen that depth estimation method is far more expensive than our proposed signature extraction; the cost for just estimating the depth can be 100 times more than our signature extraction. As a result, [12] is only a suitable solution for 2D-plus-depth formats where the expensive operation of depth estimation is not needed. Moreover, [12] uses the depth signature as a filtering step before the visual fingerprint to reduce the cost of computing visual signatures. While this argument is valid for 2D-plus-depth videos, this is not the case for stereo videos because computing the dense depth map will be more expensive than the visual signature.

Our results show that a coarse-grained disparity map is robust against multiple kinds of transformations without compromising the precision, which suggests that computing the dense depth map is not needed for such a problem.

Effect of Frame Sampling on Performance. In order to speed up the matching process, simple techniques such as frame sampling can be used. With a sampling rate of $1/n$, only one frame every n frames will get processed, therefore the matching process will get n times faster. In order to ensure the robustness of our system when using frame sampling, we repeated the individual transformations experiments with different sampling rates. The figures are omitted due to space limitations. Our results show that our system can achieve high accuracy even with low sampling rates, although the recall drops slightly with the decrease of sampling rate, because of the reduced amount of data. For example, we can achieve 92% recall at 96% precision with a sampling rate of $1/10$, while without frame sampling we achieve a recall of 96% at 96% precision. That is, there is a loss of 4% in recall with a rate of $1/10$. Also, with a sampling rate of $1/25$ (about one frame per sec), we can achieve 86% recall at 92% precision, while without frame sampling the achieved recall at precision 92% is 96%. Thus, there is only a loss of 10% in recall with sampling rate of $1/25$. As a result, we can speed up the matching process significantly, while still having high accuracy.

D. Accuracy and Scalability of the Matching Engine

We evaluate the accuracy and scalability of the matching engine component of the proposed system. We also compare our matching engine versus the best results reported by the RankReduce system [22], which is the closest to our work.

Accuracy and Comparison Against RankReduce. We focus on evaluating the accuracy of the computed nearest neighbors, which is the primitive function provided by the engine. The accuracy of the retrieved K nearest neighbors for a point p is computed using the $Precision@K(p)$ metric, which is given by:

$$Precision@K(p) = \frac{\sum_{i=1}^{K}\{T_i <= K\}}{K}, \tag{5}$$

where T_i is the rank of a true neighbor. $T_i <= K$ equals 1 if a true neighbor is within the retrieved K, and 0 otherwise. The average precision of the retrieved K nearest neighbors across all points in the query set Q is:

$$AvgPrecision@K = \frac{\sum_{i=1}^{|Q|}\{Precision@K(i)\}}{|Q|}. \tag{6}$$

We use the $AvgPrecision@K$ metric in our experiments.

We compare against RankReduce [22], which implements a distributed LSH index. It maintains a number of hash tables over a set of machines on a distributed file system, and it uses MapReduce for

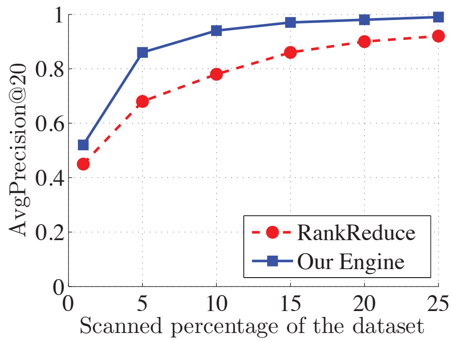


Fig. 5. Comparing our matching engine versus the closest system in the literature, RankReduce.

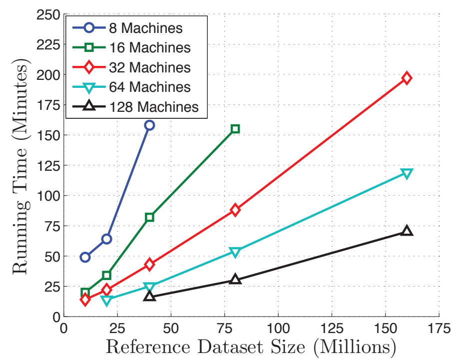


Fig. 6. Running times of different dataset sizes on different number of machines.

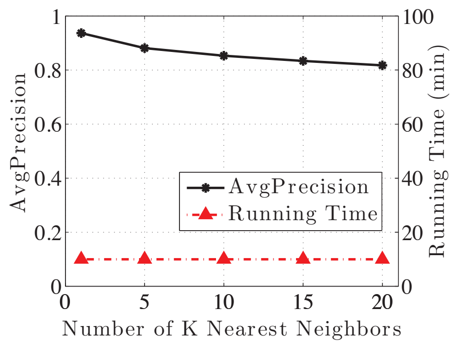


Fig. 7. The effect of K on running time and accuracy.

searching the tables for similar points. We compare the results achieved by our matching engine against the *best* results mentioned in [22] using the same dataset and the same settings. We did not implement RankReduce; rather we use the the best stated results in its paper. We use the same dataset size of 32,000 points extracted from visual features of images. We measure the average precision at 20 nearest neighbors at the same percentage of scanned bins, which are called probed buckets in RankReduce terms.

We plot the comparison results in Figure 5. The results first show that the proposed matching engine produces high accuracy, which is more than 95% by scanning less than 10% of the data. In addition, the results show that our matching engine consistently outperforms RankReduce, and the gain is significant (15–20%) especially in the practical settings when we scan 5–10% of the data points. For example, when the fraction of scanned data points is 5%, the average precision achieved by our engine is about 84%, while the average precision achieved by RankReduce is less than 65% for the same fraction of scanned data points. For RankReduce to achieve 84% average precision, it needs to scan at least 15% of the dataset (3X more than our engine), which incurs significantly more computation and I/O overheads.

In addition to the superior performance in terms of average precision, our engine is more efficient in terms of storage and computation. For storage, RankReduce needs to store the whole reference dataset multiple times in hash tables; up to 32 times. On the other hand, our engine stores the reference dataset only once in bins. Storage requirements for a dataset of size 32,000 points indicate that RankReduce needs up to 8 GB of storage, while our engine needs up to 5 MB, which is more than 3 orders of magnitude less. These storage requirements may render RankReduce not applicable for large datasets with millions of points, while our engine can scale well to support massive datasets. For computation resources, our engine and RankReduce use similar scan method to reference points found in bins or buckets. However, as discussed above, RankReduce needs to scan more buckets to produce similar precision as our engine. This makes our engine more computationally efficient for a given target precision, as it scans fewer bins.

Scalability and Elasticity of our Engine. We conduct multiple experiments to show that our engine is scalable and elastic. Scalability means the ability to process large volumes of data, while elasticity indicates the ability to efficiently utilize various amounts of computing resources. Both are important characteristics: scalability is needed to keep up with the continuously increasing volumes of data and elasticity is quite useful in cloud computing settings where computing resources can be acquired on demand.

We run our engine on datasets of different sizes from 10 to 160 million data points, and on clusters of sizes ranging from 8 to 128 machines from Amazon. These data points are visual features extracted from 1 million images download from ImageNet [8]. From each image, we extract up to 200 SIFT features, which results in a dataset of up to 200 million data points. In all experiments, we compute the $K = 10$ nearest neighbors for a query dataset of size 100,000 data points. We measure the total running time to complete processing all queries, and we plot the results in Figure 6. The figure shows that our engine is able to handle large datasets, up to 160 million reference data points are used in creating the distributed index. More importantly, the running time grows almost linearly with increasing the dataset size on the same number of machines. Consider for example the curve showing the running times on 32 machines. The running times for the reference dataset of sizes 40, 80, and 160 million data points are about 40, 85, and 190 minutes, respectively.

In addition, the results in Figure 6 clearly indicate that our engine can efficiently utilize any available computing resources. This is shown by the almost linear reduction in the running time of processing the same dataset with more machines. For example, the running times of processing a reference dataset of

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TMM.2015.2389628, IEEE Transactions on Multimedia

size 80 million data points are 160, 85, 52, and 27 minutes for clusters of sizes 16, 32, 64, and 128 machines, respectively. The scalability and elasticity of our engine are obtained mainly by our design of the distributed index, which partitions the datasets into independent and non-overlapping bins. These bins are allocated independently to computing machines for further processing. This data partitioning and allocation to bins enable flexible and dynamic distribution of the computational workload to the available computing resources, which is supported by the MapReduce framework.

Effect of Number of K Nearest Neighbors. In this experiment, we study the effect of changing the number of K nearest neighbors retrieved. We measure the running time and the average precision for different values of K, while maintaining a fixed scanned percentage of the reference dataset. The results are plotted in Figure 7, which show that while we achieve high precision for returning the closest neighbor (i.e., $K = 1$), with value of 94%, the average precision achieved is not significantly impacted by increasing K. For example at $K = 5$ the average precision is 88%, and at $K = 20$ the average precision is 82%, losing only 6% of the precision when returning 4 times more neighbors. The results also show that the effect of K on running time is negligible, since running time is mainly related to the size of the scanned data points.

VII. Conclusions and Future Work

Distributing copyrighted multimedia objects by uploading them to online hosting sites such as YouTube can result in significant loss of revenues for content creators. Systems needed to find illegal copies of multimedia objects are complex and large scale. In this paper, we presented a new design for multimedia content protection systems using multi-cloud infrastructures. The proposed system supports different multimedia content types and it can be deployed on private and/or public clouds. Two key components of the proposed system are presented. The first one is a new method for creating signatures of 3D videos. Our method constructs coarse-grained disparity maps using stereo correspondence for a sparse set of points in the image. Thus, it captures the depth signal of the 3D video, without explicitly computing the exact depth map, which is computationally expensive. Our experiments showed that the proposed 3D signature produces high accuracy in terms of both precision and recall and it is robust to many video transformations including new ones that are specific to 3D videos such as synthesizing new views. The second key component in our system is the distributed index, which is used to match multimedia objects characterized by high dimensions. The distributed index is implemented using the MapReduce framework and our experiments showed that it can elastically utilize varying amount of computing resources and it produces high accuracy. The experiments also showed that it outperforms the closest system in the

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TMM.2015.2389628, IEEE Transactions on Multimedia

literature in terms of accuracy and computational efficiency. In addition, we evaluated the whole content protection system with more than 11,000 3D videos and the results showed the scalability and accuracy of the proposed system. Finally, we compared our system against the Content ID system used by YouTube. Our results showed that: (i) there is a need for designing robust signatures for 3D videos since the current system used by the leading company in the industry fails to detect most modified 3D copies, and (ii) our proposed 3D signature method can fill this gap, because it is robust to many 2D and 3D video transformations.

The work in this paper can be extended in multiple directions. For example, our current system is optimized for batch processing. Thus, it may not be suitable for online detection of illegally distributed multimedia streams of live events such as soccer games. In live events, only small segments of the video are available and immediate detection of copyright infringement is crucial to minimize financial losses. To support online detection, the matching engine of our system needs to be implemented using a distributed programming framework that supports online processing, such as Spark. In addition, composite signature schemes that combine multiple modalities may be needed to quickly identify short video segments. Furthermore, the crawler component needs to be customized to find online sites that offer pirated video streams and obtain segments of these streams for checking against reference streams, for which the signatures would also need to be generated online. Another future direction for the work in this paper is to design signatures for recent and complex formats of 3D videos such as multiview plus depth. A multiview plus depth video has multiple texture and depth components, which allow users to view a scene from different angles. Signatures for such videos would need to capture this complexity, while being efficient to compute, compare, and store.

References

[1] A. Abdelsadek. Distributed index for matching multimedia objects. Master's thesis, Simon Fraser University, Canada, 2014.

[2] A. Abdelsadek and M. Hefeeda. Dimo: Distributed index for matching multimedia objects using mapreduce. In *Proc. of ACM Multimedia Systems Conference (MMSys'14)*, pages 115–125, Singapore, March 2014.

[3] M. Aly, M. Munich, and P. Perona. Distributed Kd-Trees for Retrieval from Very Large Image Collections. In *Proc. of British Machine Vision Conference (BMVC)*, Dundee, UK, August 2011.

[4] J. Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517, September 1975.

[5] P. Cano, E. Batle, T. Kalker, and J. Haitsma. A review of algorithms for audio fingerprinting. In *Proc. of IEEE Workshop on Multimedia Signal Processing*, pages 169–173, St. Thomas, US Virgin Islands, December 2002.

[6] Copyright on Youtube. http://www.youtube.com/yt/copyright/.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TMM.2015.2389628, IEEE Transactions on Multimedia

[7] J. Dean and S. Ghemawat. Mapreduce: simplified data processing on large clusters. In *Proc. of Symposium on Operating Systems Design and Implementation (OSDI'04)*, pages 137–150, San Francisco, CA, December 2004.

[8] J. Deng, W. Dong, R. Socher, L. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Proc. of IEEE Conference on Computer Vision and Pattern Recognition (CVPR'09)*, pages 248–255, Miami, FL, June 2009.

[9] A. Hampapur, K. Hyun, and R. Bolle. Comparison of sequence matching techniques for video copy detection. In *Proc. of SPIE Conference on Storage and Retrieval for Media Databases (SPIE'02)*, pages 194–201, San Jose, CA, January 2002.

[10] S. Ioffe. Full-length video fingerprinting. Google Inc., July 24 2012. US Patent 8229219.

[11] A. Kahng, J. Lach, W. Mangione-Smith, S. Mantik, I. Markov, M. Potkonjak, P. Tucker, H. Wang, and G. Wolfe. Watermarking techniques for intellectual property protection. In *Proc. of the 35th Annual Design Automation Conference (DAC'98)*, pages 776–781, San Francisco, CA, June 1998.

[12] N. Khodabakhshi and M. Hefeeda. Spider: A system for finding 3d video copies. *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)*, 9(1):7:1–7:20, February 2013.

[13] S. Lee and C. Yoo. Robust video fingerprinting for content-based video identification. *IEEE Transactions on Circuits and Systems for Video Technology*, 18(7):983–988, July 2008.

[14] H. Liao, J. Han, and J. Fang. Multi-dimensional index on hadoop distributed file system. In *Proc. of IEEE Conference on Networking, Architecture and Storage (NAS'10)*, pages 240–249, Macau, China, July 2010.

[15] Z. Liu, T. Liu, D. Gibbon, and B. Shahraray. Effective and scalable video copy detection. In *Proc. of ACM Conference on Multimedia Information Retrieval (MIR'10)*, pages 119–128, Philadelphia, PA, March 2010.

[16] J. Lu. Video fingerprinting for copy identification: From research to industry applications. In *IS&T/SPIE Electronic Imaging*, volume 7254 of *SPIE Proceedings*, pages 725402–725402. International Society for Optics and Photonics, SPIE, 2009.

[17] W. Lu, Y. Shen, S. Chen, and B. Ooi. Efficient processing of k nearest neighbor joins using mapreduce. *Proceedings of the VLDB Endowment (PVLDB)*, 5(10):1016–1027, June 2012.

[18] E. Metois, M. Shull, and J. Wolosewicz. Detecting online abuse in images. Markmonitor Inc., Apr. 12 2011. US Patent 7925044.

[19] H. Müller, W. Müller, D. Squire, S. Marchand-Maillet, and T. Pun. Performance evaluation in content-based image retrieval: overview and proposals. *Pattern Recognition Letters*, 22(5):593–601, April 2001.

[20] P. Ram and A. Gray. Which space partitioning tree to use for search? In *Proc. of Advances in Neural Information Processing Systems (NIPS'13)*, pages 656–664, Lake Tahoe, NV, December 2013.

[21] V. Ramachandra, M. Zwicker, and T. Nguyen. 3D video fingerprinting. In *Proc. of 3DTV Conference: The True Vision - Capture, Transmission and Display of 3D Video (3DTV'08)*, pages 81 –84, Istanbul, Turkey, May 2008.

[22] A. Stupar, S. Michel, and R. Schenkel. Rankreduce - processing k-nearest neighbor queries on top of mapreduce. In *Proc. of Workshop on Large-Scale Distributed Systems for Information Retrieval (LSDS-IR'10)*, pages 13–18, Geneva, Switzerland, July 2010.

[23] K. Tasdemir and A. Cetin. Motion vector based features for content based video copy detection. In *Proc. of International Conference on Pattern Recognition (ICPR'10)*, pages 3134 –3137, Istanbul, Turkey, August 2010.

[24] U. Capeto. Depth Map Automatic Generator. http://3dstereophoto.blogspot.com/2013/04/depth-map-automatic-generator-dmag.html, April 2013.

[25] Vobile Launches VDNA 3D. http://www.vobileinc.com/press.php?id=23.

[26] ISO/IEC JTC1/SC29/WG11, Reference softwares for depth estimation and view synthesis. Doc. M15377, April 2008.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TMM.2015.2389628, IEEE Transactions on Multimedia

Mohamed Hefeeda received his Ph.D. from Purdue University, USA in 2004, and M.Sc. and B.Sc. from Mansoura University, Egypt in 1997 and 1994, respectively. He is a Principal Scientist in the Qatar Computing Research Institute (QCRI), Doha, Qatar. He is on leave from Simon Fraser University, Canada, where he is a Professor in the School of Computing Science. His research interests include multimedia networking over wired and wireless networks, peer-to-peer systems, mobile multimedia, and cloud computing. In 2011, he was awarded one of the prestigious NSERC Discovery Accelerator Supplements (DAS), which were granted to a selected group of researchers in all Science and Engineering disciplines in Canada. His research on efficient video streaming to mobile devices has been featured in multiple international news venues, including ACM Tech News, World Journal News, and CTV British Columbia. He serves on the editorial boards of several premier journals such as the ACM Transactions on Multimedia Computing, Communications and Applications (TOMM), where he was named the best associate editor in 2014. He has served on many technical program committees of major conferences in his research area, such as ACM Multimedia. Dr. Hefeeda has co-authored more than 80 refereed journal and conference papers and has two granted patents.

Tarek ElGamal received the B.Sc. degree from Cairo University, Egypt in 2011. He is working as a software engineer in the Qatar Computing Research Institute (QCRI), Doha, Qatar. Prior to joining QCRI, he was a software engineer in the Microsoft Advanced Technology Labs in Cairo (ATLC). His research interests are in the areas of cloud computing, distributed systems, and big data analytics.

Kiana Calagari received the BSc and MSc degrees in Electrical Engineering from Sharif University of Technology, Iran, in 2010 and 2012, respectively. She is currently a PhD student in Computing Science at Simon Fraser University, Canada. Her research interests include multimedia systems, multimedia networking and cloud computing.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TMM.2015.2389628, IEEE Transactions on Multimedia

Ahmed Abdelsadek received the BSc degree in Computing Science from the Faculty of Computers and Information, Cairo University, Egypt in 2010, and the MSc degree in Computing Science from Simon Fraser University, Canada in 2014. His research interests include Distributed Systems and Content-based Multimedia Information Retrieval.