Optimizing Near Duplicate Detection for P2P Networks

Odysseas Papapetrou, Sukriti Ramesh, Stefan Siersdorfer, Wolfgang Nejdl
L3S Research Center, Hannover, Germany
Email: {papapetrou, ramesh, siersdorfer, nejdl}@l3s.de

***Abstract*—In this paper, we propose a probabilistic algorithm for detecting near duplicate text, audio, and video resources efficiently and effectively in large-scale P2P systems. To this end, we present a thorough cost and probabilistic analysis that allows the algorithm to adapt to network and data collection characteristics for minimizing network cost. In addition, we extend the algorithm so that it can identify similar videos, even if some of the videos are split into different files. A thorough theoretical analysis as well as a large-scale experimental evaluation on networks of up to 100,000 peers using real-world datasets of more than 200 Gbytes demonstrate the viability of our approach.**

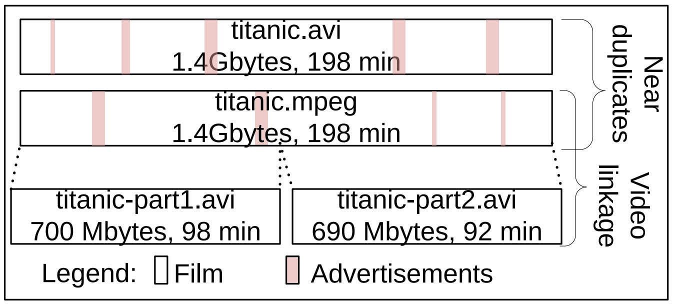

Fig. 1. Near duplicates and video linkage.

I. Introduction

Efficiently and effectively searching for very similar files over large file repositories is an active research topic, and was extended for handling file types beyond text, e.g., video [22], audio [20], and images [10]. This problem is generally known as *Near Duplicate Detection (NDD)*. Algorithms for NDD have various applications in large file repositories such as reduction of storage requirements [23], and detection of copyrighted multimedia content [19], [16].

Near duplicates are frequently created in P2P networks during normal file sharing operations, an example being different recordings of the same movie that may exist concurrently in a P2P file sharing network (cf. Fig. 1) whereby minor differences between recordings can occur, e.g., due to advertisements, or lossy compression. NDD can be used to find multiple sources/peers for downloading the same resource in parallel, to find the same video or audio resource at higher resolution, or to filter out near duplicates from query results in order to present only novel results to a peer. P2P networks that focus on Multimedia Information Retrieval, e.g., SAPIR (www.sapir.eu) and VICTORY (www.victory-eu.org), also benefit from identifying near duplicates for enabling content-based retrieval of videos and audio files [6]. However, NDD methods implemented in these systems cannot scale to large P2P networks, since they assume a central repository, and are based on costly, pair-wise comparisons to detect the near duplicates.

In addition to centralized NDD algorithms, there are recent proposals for P2P and distributed NDD algorithms, e.g., [1], [9], [6], [20]. Most of these employ a family of algorithms called *Locality Sensitive Hashing (LSH)* [3], [8] to map resources to bit strings, and to build an index suitable for efficiently answering *K-Nearest Neighbor (KNN)* queries. NDD queries can be reduced to incremental KNN queries, e.g., keep querying for nearest neighbors until the difference threshold for near duplicates is surpassed. Although these works achieve good results in terms of effectiveness, they can be substantially improved in terms of efficiency by considering network and collection characteristics, which is the main objective of our paper.

In contrast to previous works, in this paper we address NDD directly, instead of considering it as a special case of KNN. This allows us to optimally select tuning parameters of LSH which affect the network cost and the probabilistic guarantees for the detection of the near duplicates. *In this work, by tuning these parameters, we minimize the network cost for LSH, and we enable querying with constant cost and with probabilistic guarantees, both of which are not possible with existing P2P algorithms.* Our approach, called **POND** (short for Peer-to-peer Optimized Near Duplicate detection), is both efficient and effective, and applicable to large P2P networks and to a variety of file types. In our experiments, the optimization of POND results in a cost reduction of several orders of magnitude.

In addition, we extend POND to cover a particular requirement of modern file sharing P2P networks: to link together all videos that have large near duplicate segments (e.g., "titanic.mpeg", with "titanic-part1.avi" and "titanic-part2.avi" in Fig. 1), a problem known as *Video Linkage (VL)* [11]. This video splitting occurs frequently, e.g., when burning large videos to CDs. By linking together these videos, we can parallelize further the downloading of large video files, and enable recovery of the downloading process even if all sources of the original video file are disconnected.

The rest of this paper is organized as follows: We discuss existing work on NDD in Section II. In Section III we present the preliminaries for near duplicate detection. We describe the P2P infrastructure used by POND in Section IV. Our main contribution, the adaptation of POND to the network and collection characteristics for minimizing the network usage under probabilistic guarantees, is presented in Section V. Section VI presents a large-scale experimental evaluation of

POND, using a real-world collection of over 200 Gbytes, consisting of videos, audio, and text. We conclude and show directions of our future work in Section VII.

II. Related work

Locality Sensitive Hashing (LSH) [8] has recently received substantial attention with respect to the K-Nearest Neighbor (KNN) problem, as well as for Near Duplicate Detection (NDD). The main idea behind LSH, reviewed in Section III, is to map similar elements to the same hash value with high probability (using distance thresholds for both "high" and "low" similarity). Bawa et al. [1] propose LSH Forest, which allows the distance thresholds to be set per query, and show how it can be deployed in a P2P network. However, LSH Forest does not optimize network usage for NDD queries; since its purpose is to enable the thresholds to be set per query, it cannot tune certain parameters of LSH (more specifically, number of hash tables l and number of hash functions k described later in Section III) which, as we will show, are crucial for minimizing the total network cost. In fact, the cost of indexing each resource in LSH Forest is $O(l \cdot k)$, which is even higher than the corresponding cost of standard LSH ($O(\log(l))$ [8]. Furthermore, for query execution, the number of peers that need to be visited is not constant as in the standard LSH, but varies depending on the query parameters. In this work we have a different focus than LSH Forest. For the applications we consider and describe in Section I, the distance thresholds do not change often; the thresholds may still change at runtime, but not necessarily per query. Having fixed thresholds enables us to select values for k and l that minimize the network cost, and to save a substantial amount of network resources.

Similar to LSH Forest, several other P2P approaches address the NDD problem by using a KNN infrastructure. In a recent work [9], Haghani et al. present a P2P system based on LSH, and in particular on p-stable hashing. Their approach significantly reduces the cost compared to LSH Forest, as it requires only l indexes per resource. However, similar to LSH Forest, this work focuses on KNN queries. As such, even though it can execute near duplicate queries by reducing them to incremental KNN queries, it does not optimize the LSH configuration for minimizing the network cost. Due to this lack of optimization, the cost for answering a NDD query is not constant. By restricting the problem domain to NDD queries only, the analysis and optimization presented in our work could also be applied to their system for optimizing the usage of network resources.

Dong et al. [4] focus on deriving analytically the optimal configuration for LSH with the objective of minimizing the computational time and maximizing the recall. They show significant performance increase compared to an unoptimized LSH. However, their approach only considers centralized scenarios for which the expensive resource is the computation time. As such, they do not consider the network usage which is an important resource for high-churn P2P networks like the ones considered in this work. Therefore, their results are not applicable to P2P.

There is also some work on NDD and KNN for P2P systems not based on LSH. Falchi et al. [6] present DINN, an algorithm for incremental nearest neighbor in P2P based on priority queues, which however requires an efficient P2P range query, or an incremental P2P NDD implementation. Otherwise, DINN needs to route each query to almost all peers, and then cannot scale. Yang [20] proposes a P2P version of MACSIS, an algorithm for detecting near duplicate audio files. The algorithm is built over unstructured P2P networks, and uses gossiping for query execution; therefore, it cannot scale to more than a few hundred peers.

To the best of our knowledge, our paper is the first to propose a scalable P2P algorithm that adapts to the peer contents, the network size, and the desired probabilistic guarantees for addressing the NDD problem with near-optimal network cost, without overloading the participating peers.

III. Prerequisites on Near Duplicate Detection

In this section, we will provide an overview on Near Duplicate Detection. To this end, we first formalize the objectives of near duplicate detection and video linkage. We then discuss representations for different multimedia resources, and finally describe Locality Sensitive Hashing (LSH) which forms the technical basis of POND.

A. Problem Definition

Each resource $x \in \mathcal{X}$ has a *type*, e.g., audio, video, text. Resources of the same type can have different formats, e.g., an audio resource can be mp3, wav, etc. A *resource representation* $R(x)$ is a normalized format-independent representation of resource x. In this work, we represent resources as sets of strings. The transformation which produces $R(x)$ from x depends on the specific resource type. We will summarize different suitable representations and transformations for video, audio, and text files in Section III-B.

Definition 1 (Similarity function): Given two resources x_1 and x_2 of the same type $T \in \{audio, video, text\}$, the *similarity function* $Sim(R(x_1), R(x_2))$ computes the similarity of the two resources based on their resource representations. The similarity values are in the range $[0, 1]$.

A similarity value of 0 means that the resources are completely dissimilar, while a similarity value of 1 denotes identical resources. In this work we measure similarity using Jaccard similarity, which is the standard similarity function for sets: $Sim(x_1, x_2) := \frac{|R(x_1) \cap R(x_2)|}{|R(x_1) \cup R(x_2)|}$.

Definition 2 (Near Duplicate resources): Given resources x_1 and x_2 of type T. The resources are *near duplicates* under similarity threshold t if they have $Sim(R(x_1), R(x_2)) \geq t$.

Threshold t is in the range of $[0 \ldots 1]$. Its value depends on the application scenario. For instance, typical values for text are between 0.8 and 0.95 [18]. For our application scenarios, e.g., parallelizing downloads, or locating multimedia resources of different resolutions (cf. Section I), this threshold is a fairly stable system property for each resource type. For example, downloading a video in parallel from two sources works only if the two sources have a similarity higher than a high threshold, otherwise merging of the two sources will produce unwanted artifacts in the video. This threshold can even be determined a priori, according to the merging algorithm and the acceptable quality loss.

As explained in Section I, it is often the case that videos have large overlapping segments, e.g., large videos may be partitioned into two or more files for easy storage. The smaller parts may additionally undergo post-processing, like compression or re-encoding with a different encoder. We want to be able to detect the relation between the partitioned files and the original file, and link all resources together.

Definition 3 (Video Linkage): Given video resources x_1 and x_2, the videos are *linked* if there exist non-empty segments of x_1 and x_2, denoted as $Segment(x_1)$ and $Segment(x_2)$, such that $Segment(x_1)$ and $Segment(x_2)$ are near duplicates.

In this work we use 'part' to denote the physical partitioning of a video into two or more files (e.g.,titanic-part1.avi and titanic-part2.avi in Figure 1). In contrast, term 'segment' refers to a conceptual splitting of a video into smaller segments.

B. Resource Representations for NDD

POND can operate on different resource representations and similarity functions for each resource type. To keep the algorithm's description independent of the resource types, we now introduce a common resource representation for all resource types, and describe appropriate transformations to convert each resource to this common representation. We also discuss about alternative representations and similarity functions, which may be preferred for some scenarios.

Text resources. We process a text document x to produce its representation $R(x)$ as follows: (a) extract all terms from x, (b) stem all extracted terms, and (c) filter out stopwords. Representation $R(x)$ consists of the set of remaining terms. This combination of the bag-of-words representation and Jaccard similarity is frequently used in IR for computing similarity of text documents.

Audio resources. The standard approach for evaluating similarity of two audio files consists of creating textual acoustic fingerprints of the two files, and then using text-based similarity measures to compute the similarity. Two similar files are expected to have similar fingerprints. The most frequently used non-proprietary fingerprinting technique is *fooid* [12] which is integrated in several media players. Fooid is fast and portable, and produces fingerprints which are independent of the file format/encoding. For each audio file, fooid produces 424 fingerprints, one for each dimension, e.g., energy, tonality, length. We convert these fingerprints to strings by concatenating the dimension id with the dimension value for each dimension. This results in a set of 424 strings for each audio file, which we use as a representation.

Video resources. In this work we use a representation of videos based on the color histograms of the keyframes [21]. The keyframes of a video are the frames that differ significantly from their preceding frames, i.e., more than a given threshold. To generate the video representation, we first detect the video keyframes, and compute their color histograms in the HSV space. We then generate a fingerprint of each histogram by finding and sorting the k most populated buckets, and concatenating their indices in a string. The video representation consists of the set of fingerprints of all keyframe histogram signatures. Video representations based on keyframe histograms are a standard technique for partitioning and indexing video resources [21], as well as for finding near duplicate videos [17], [19].

In order to address the video linkage problem, videos are conceptually split into segments. The splitting points are decided using the keyframe extraction algorithm described earlier, but using a higher distance threshold. Each video segment is handled as an individual resource, with its own representation. A record of the video segmentations is maintained by the peer that owns the video, such that the originating video of each segment can be found during query execution. Therefore, detecting a single near duplicate segment for the query is sufficient for detecting the originating video, and for linking it with the query.

C. Locality Sensitive Hashing

Locality Sensitive Hashing is often used for NDD in centralized environments. POND is also based on LSH (specifically on [8]), which we briefly describe now. Our description follows the notation of [8].

LSH is based on the notion of locality sensitive hash families. Let $\mathcal{M}$ be a metric space (e.g., with dimensions corresponding to terms for text documents), and $d(\cdot,\cdot)$ the distance function defined for any two points of $\mathcal{M}$, (e.g., the Jaccard distance between two documents). A family of hash functions $\mathcal{H}$ is *locality sensitive* if there exist positive thresholds r_1 and r_2, and probabilities pr_1 and pr_2 for which the following conditions hold:

- **C1:** If the distance between two points $d(p,q)$ (e.g., two text documents) is less than the distance threshold r_1, then $f(p) = f(q)$ with probability at least pr_1, where $f(\cdot)$ is randomly chosen from $\mathcal{H}$.
- **C2:** If the distance between two points $d(p,q)$ is at least equal to r_2, then $f(p) = f(q)$ with probability at most pr_2, where $f(\cdot)$ is randomly chosen from $\mathcal{H}$.

We now demonstrate how LSH works using text documents as an example. The LSH algorithm is initialized by constructing l hash tables $ht_1, ht_2, \ldots, ht_l$. To each hash table it binds k hash functions $f_1(\cdot)$, $f_2(\cdot)$, $\ldots$ $f_k(\cdot)$, which are selected uniformly at random from $\mathcal{H}$. For inserting a document d in this structure, the document's representation $R(d)$ is used to compute l different labels of length k: $\mathcal{L}(d) = \{Label_1(d), Label_2(d), \ldots, Label_l(d)\}$. The algorithm computes $Label_j(d)$ that corresponds to hash table ht_j, in a bit-by-bit approach. Bit i of label $Label_j(d)$ is generated as follows (cf. Fig 2):

1) Hash function $f_i(\cdot)$ $(1 \leq i \leq k)$ corresponding to ht_j is used to hash each term from $R(d)$.

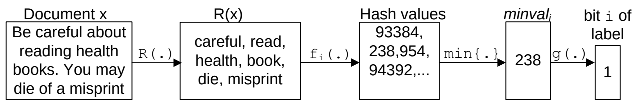
Document x
Be careful about reading health books. You may die of a misprint
R(.)
R(x)
careful, read, health, book, die, misprint
f_i(.)
Hash values
93384, 238,954, 94392,...
min{.}
minval_i
238
g(.)
bit i of label
1

Fig. 2. Computing the i bit of a label j for a text resource

2) The minimum hash value $minval_i$ produced by $f_i(\cdot)$ for all terms is detected.
3) $minval_i$ is mapped to a binary value by further hashing. The resulting bit is used as the i'th bit of document's label.

For generating all k bits for $Label_j(d)$, this process is repeated for $i = 1 \ldots k$ resulting in the set $\mathcal{L}(d)$ of all labels for d. Document d is then hashed in all l hash tables using the corresponding labels as keys.

For successful deployment of the LSH algorithm in P2P environments we require: (a) an inexpensive method to store the l hash tables in a P2P network, (b) a way to coordinate the peers so that they maintain the required hash tables, and, (c) a distributed and dynamic approach to adapt the configuration of LSH (parameters k and l) to minimize network costs. We will elaborate on these issues in the subsequent sections.

IV. POND Infrastructure

The backbone of POND is an inverted index combining Distributed Hash Tables (DHTs) [15] and Locality Sensitive Hashing (LSH) [8] described in Section III-C. POND indexes the files/resources in this inverted index such that similar resources are indexed using the same DHT keys so that they can be efficiently located with standard DHT lookups. The algorithm consists of two parts: (a) the *indexing part* (Section IV-A) which is responsible for configuring and maintaining the distributed index, and, (b) the *query execution part* (Section IV-B), responsible for efficiently locating the near duplicates of a query file.

The novelty of POND lies in the way the indexing part is adapted to the network and data collection characteristics, such that the required probabilistic guarantees are satisfied with the optimal-minimum network cost. In this section, we only sketch this process. We will describe the configuration/optimization process in detail in Section V.

A. Inverted Index Maintenance

In Section III-C we have explained how the resource labels are computed for different file types. We now elaborate on how the inverted index on resources, constructed using the resource labels as the keys, is configured and maintained. Our implementation uses Chord [15] for the inverted index, but any DHT implementation could be used.

The index maintenance algorithm is divided into three steps: (1) algorithm configuration/optimization, where the core parameters are determined for optimizing the algorithm, (2) computing the labels for each resource, and, (3) indexing the resources in the DHT to enable their retrieval with subsequent near duplicate detection queries. These steps are repeated at regular intervals so that the LSH configuration remains optimal for the network size and for the content of the peers.

Step 1: Algorithm configuration/optimization. The purpose of this step is to estimate the parameters of LSH that satisfy the probabilistic guarantees with minimum network cost. A randomly chosen peer p_i collects statistics from the network and computes the parameters that minimize the cost of POND: the optimal number l of labels per resource, and the length k of each label. Then, p_i uses the DHT overlay to broadcast k and l to all participating peers. Step 1 is a key-step which we will describe in detail in Section V.

Step 2: Computing the labels. After receiving the updated l and k values, the participating peers compute the labels for their resources – l labels of k bits for each resource. Computation of labels is an inexpensive local process, described already in Section III-C. In addition, labels are cached and reused, even if they are computed for different k and l values. So, if either k or l is increased, only the difference between the existing labels and the new labels needs to be computed.

Step 3: Indexing. After each peer has computed the labels for its resources, it needs to index each resource in the DHT-based inverted index. For each resource x and for each label $Label_i(x)$, the peer publishes a triple $T :< i, IP, RecID >$ using $Label_i(x)$ as a key, where i is a label index, IP is the IP address of the peer, and $RecID$ is a resource ID for resource x. (It is important to include i because otherwise the peer holding the triple cannot distinguish whether a resource x and a query q have the same i'th label, or whether $Label_i(x) = Label_j(q)$ with $i \neq j$. Clearly, the latter case does not imply anything about the similarity of x and q.)

To handle *churn*, peers use periodic republishing. When a peer publishes a triple in the DHT, it attaches an expiry time. If the triple is not updated within this expiry time, it is automatically removed from the DHT. The expiry time of a triple comes shortly after the next expected republishing time; thereby resources are always indexed in the DHT, and obsolete resources are removed soon after they expire.

Note that the presented infrastructure is not the only possible infrastructure for POND. The main contribution of this paper are the optimization techniques described in Section V. These techniques can be applied, with some modifications, to other systems as well, e.g., [9], [1] (see Section II).

B. Querying for near duplicate resources

A peer p_q detects the near duplicates for a query resource q as follows. First it uses $R(q)$, the representation of q, to compute the l labels of q, denoted as $\mathcal{L}(q)$. For each label $Label_i(q) \in \mathcal{L}(q)$, the peer retrieves from the DHT inverted index the triples of all resources published using the same label as a key. These resources are the candidate near duplicates. Following, p_q sends $R(q)$ to all peers that hold each of the returned resources, which then respond with links to all their near duplicate resources. In the response, only the resources that satisfy the minimum similarity with the query representation are included. As we will show in Section V, this process guarantees that each near duplicate resource will be detected with a probability larger than a configurable value pr_{min}.

N	Number of peers
$avgRes$	Average number of resources per peer
$minSim$	Minimum similarity for considering two resources as near duplicates
pr_{min}	Minimum probability that POND will detect each near duplicate resource
l	Number of labels per resource
k	Length of each label in bits

TABLE I
NOTATIONS

The same algorithm is used for querying for video linkages, with the same probabilistic guarantees. Recall from Section III-B that for VL, peers conceptually break each video to segments, and index each segment individually. With respect to query execution for video linkage, the query video is broken into segments as well, and all near duplicates are retrieved for each segment. We will show in Section VI that this approach is very effective. In practice, not all segments need to be queried; a peer stops querying as soon as it finds enough linkages for efficiently parallelizing the download.

The most important application of NDD and VL in P2P is parallelization of the downloads of large multimedia resources. Clearly, a peer that needs to find the near duplicates does not yet have the full resource, and hence it cannot compute the resource representation. Therefore, it first downloads the resource representation from another peer that has the full resource. In this work we assume that peers already have an approach to find at least one copy of the file, for instance, using keyword search over a DHT index, as LimeWire and other mainstream applications currently do. Since resource representations are very compact, even for large videos, the additional cost imposed by this process is negligible.

V. CONFIGURATION AND OPTIMIZATION OF POND

Having described the overall framework of the POND algorithm in Section IV, we now elaborate on the first indexing step of POND, i.e., configuring and optimizing parameters of POND. We first provide an overview on this step in Section V-A. In Sections V-B and V-C we present the theoretical analysis (network cost analysis and probabilistic analysis) which drives the network optimization of POND. Finally, in Section V-D we combine cost and probabilistic analysis to derive the optimal parameters l and k for LSH that minimize the total network cost and satisfy the desired probabilistic guarantees. Table I introduces the notation used in these sections.

A. Overview on the Parameter Optimization Procedure

The step of optimizing the LSH parameters l and k in POND is executed at regular intervals on randomly selected peers. The problems that need to be addressed are: (a) Random selection of a peer for executing the step, with minimal coordination between the peers, (b) Statistics gathering, (c) Computation of the optimal values of l and k and their dissemination in all participating peers.

Random Selection. For the random selection problem, we require that one peer is randomly selected for executing the algorithm every m minutes, and that all peers have the same probability of being selected. We address this problem using a simple randomized algorithm which is resistant to churn and requires no coordination. More specifically, every m minutes, each peer decides with probability $1/N$ to execute the optimization, where N denotes the overall number of peers. Thus, the expected number of algorithm executions per m minutes is one.

Statistics Gathering. After a peer is selected to execute the optimization, it estimates the following statistics:

1) the number of peers in network, using the approach proposed in [7]
2) the number of resources per peer, and the query rate
3) the probability distribution function (PDF) for all pair-wise resource similarities in the corpus

The latter two are estimated using random sampling on a small percentage of peers, 1% in our experiments. Sampling requires a small number of messages and negligible transfer volume, thereby it does not overload the participating peers. This simple method already provides good results, as verified in our experimental evaluation, but alternative methods for disseminating the statistics based on gossiping or random walks are also possible, e.g., [7].

Computation and dissemination of the optimal $\mathbf{l}, \mathbf{k}$. Using the described statistics, the peer computes the optimal values for l and k, as we will describe in detail in Section V-D. The optimal configuration is then disseminated in the network, such that it can be further used for index maintenance and query execution. For disseminating the configuration efficiently, the peer constructs a message including the collected statistics and the estimated optimal values for k and l. The message is tagged with the local peer time, and broadcasted over the DHT using an approach proposed by El-Ansary et al. [5]. The algorithm requires $O(N)$ messages and $O(\log(N))$ time. It might happen that two peers are selected to perform parameter optimization almost simultaneously. This is not a problem for POND since during configuration dissemination only the most recent configuration is kept.

B. Cost analysis

The network cost of POND is composed as follows: (a) the cost of maintaining the DHT overlay, (b) the cost of maintaining the locality-sensitive inverted index over DHT, and (c) the cost of querying for near duplicates. The cost of maintaining the DHT overlay is orthogonal to the POND algorithm, therefore we do not integrate it in our analysis. The reader can find a detailed cost analysis for Chord in [15].

Inverted index maintenance cost. Indexing one resource in the DHT requires at most $l \times (\log(N) + 1)$ messages:

- Executing l DHT lookups (one DHT lookup for each of the resource labels) requires at most $l \times \log(N)$ messages.
- Using the l labels as keys, and publishing the resource's meta-data in the DHT requires l additional messages.

Thus, the total number of messages for indexing all resources in the network is: $C_{maint} \leq N \cdot avgRes \cdot (l \cdot \log(N) + l) = O(N \cdot avgRes \cdot l \cdot \log(N))$.

Query execution cost. Let q denote the resource for which we wish to find the near duplicates, $ND(q)$ the set of all near duplicates for q, and $FP(q)$ the set of resources that are falsely identified as near duplicates (the false positives). Finding the near duplicates of q requires $C_{find} \leq l \cdot (\log(N) + 2) + 2 \cdot |ND(q)| + |FP(q)|$ messages:

- Executing a DHT lookup for each of the l labels of q requires a maximum of $l \cdot \log(N)$ messages.
- Retrieving the candidate resources for the l labels requires $2 \cdot l$ additional messages.
- Sending $R(q)$, the representation of the query, to the peers holding all candidate near duplicates requires a maximum of $|ND(q)| + |FP(q)|$ messages.
- Retrieving all near duplicate resources requires a maximum of $|ND(q)|$ messages.

The total cost per republishing period is a linear combination of C_{maint} and C_{find}. Let y denote the expected number of queries at each republishing period. Then, the total cost is $C_{total} = C_{maint} + y \cdot C_{find}$.

We now have the cost expressions for all aspects of the algorithm. In the following sections we show: (a) how to get an estimate for $|FP(q)|$, and, (b) how to choose the two parameters of POND, l and k, such that the total cost is minimized.

C. Probabilistic Analysis

First, we compute the probability that a near duplicate resource will be detected by POND. Then, we derive the expectation for the number of false positives that the algorithm retrieves per query.

The following lemma computes the probability that two resources x and y have the same label $Label_j$, corresponding to hash table ht_j.

Lemma 4: Given resources x and y with similarity $Sim(x,y)$. The probability that the j'th label of x is identical to the j'th label of y is:

$$Pr\left[Label_j(x) = Label_j(y)\right] = \sum_{i=0}^{k} \binom{k}{i} \cdot (1 - Sim(x,y))^i \cdot Sim(x,y)^{(k-i)} \cdot (0.5)^i$$

Proof: $Pr\left[Label_j(x) = Label_j(y)\right]$ can be expressed as a product of the individual probabilities of the k bits in the corresponding labels to match. As explained in the previous section, the individual bits of the labels are set using min-wise hashing followed by binary hashing. If the result of min-wise hashing of two resources is the same, the result of the binary hashing is also assured to be the same. If the result of the min-wise hashing is not the same, the result of binary hashing can still be the same with probability 0.5. More specifically, the probability that x and y have the same label $Label_j$ is computed as follows. Consider the min-wise hashing value computed for a single hash function $f_i(\cdot) \in \mathcal{H}_j$. The probability that min-wise hashing of the two resources using hash function $f_i(\cdot)$ yields the same result depends on the similarity of the resources. For the case that $Sim(x,y)$ denotes Jaccard similarity between the two resources, Broder et al. [2] show the following:

$$Pr[\min\{f_i(x)\} = \min\{f_i(y)\}] = Sim(x,y) \qquad (1)$$

where $\min\{f_i(\cdot)\}$ denotes the min-wise hashing values of the resources using f_i. Because of pairwise independence of the hash functions in $\mathcal{H}_j$, the probability that i of the k min-wise hash values of x and y are pairwise equal is $Sim(x,y)^i$. It follows that the remaining $(k-i)$ min-wise hash values are not pairwise equal with a probability of $(1 - Sim(x,y))^{k-i}$. Owing to binary hashing that follows the min-wise hashing, the probability of these $(k-i)$ min-hash values to still map to equal binary values is given by $((1-Sim(x,y))^{k-i} \cdot (0.5)^{k-i})$. Factor $\binom{k}{i}$ accounts for all combinations of i out of k ($0 \leq i \leq k$). ■

Clearly, we cannot consider $Sim(x,y)$ for all possible resource pairs. However, by definition the similarity of two near duplicate resources is at least $minSim$, and we can use that to obtain a lower bound for the probability in Lemma 4. We now compute the probability that two resources x and y have at least one common label.

Theorem 5: Given resources x and y with similarity $Sim(x,y)$. The probability that x and y have at least one common corresponding label is given by

$$pr_{\text{found}} = 1 - (1 - Pr\left[Label_j(x) = Label_j(y)\right])^l \qquad (2)$$

for any $j \in [1 \ldots l]$.

Proof: The j'th label of x and y do not match with a probability of $(1 - Pr[Label_j(x) = Label_j(y)])$, which can be computed with Lemma 4. The probability that none of the labels of x and y match is $(1-Pr\left[Label_j(x) = Label_j(y)\right])^l$. The probability that at least one of the l labels matches is given by $pr_{\text{found}} = 1 - (1 - Pr\left[Label_j(x) = Label_j(y)\right])^l$. ■

The POND algorithm will detect two resources as near duplicates if they have at least one common corresponding label. Therefore, the probability that each near duplicate will be detected by the algorithm corresponds to the probability given by Eqn. 2.

Estimating the number of false positives. The false positives for a query q are the resources that are detected by the algorithm as near duplicates but have similarity with q lower than $minSim$. Peers in POND detect these resources as false positives before transferring them over the network. Nevertheless, as explained earlier, false positives cause additional network overhead, due to the process involved for detecting and filtering them out. Therefore, we need to estimate the number of false positives for the purpose of accurately modeling and minimizing the network cost.

We use $\mathcal{S}$ to denote the set of all resources in the network having similarity with q less than $minSim$. Using Theorem 5, the expected number of false positives $E(|FP(q)|)$ can be computed as follows:

$$E(|FP(q)|) = \sum_{x \in \mathcal{S}} (1 - (1 - Pr\left[Label_j(x) = Label_j(q)\right])^l)$$

The estimation of the number of false positives using the above equation is costly to compute as it requires computing all similarities between the query and all available resources. An efficient alternative is to estimate $|FP(q)|$ using an approximation for the probability distribution of similarities. Let

$p(x)$ denote the probability distribution function *(PDF)* of the pairwise similarities of all resources in the corpus. Then, the expected number of false positives is:

$$E(|FP(q)|) = avgRes \cdot N \cdot \int_0^{minSim} p(x) \cdot pr_{labels}(x)\ dx$$

where $pr_{labels}(x)$ denotes the probability that two resources with similarity x share at least one label, and is computed using Equation 2. For a discrete PDF, we derive an analogous expression by replacing the integral with a sum.

The PDF $p(x)$ depends on the corpus collection and the chosen similarity measures. Our experiments with large collections (Section VI) indicate that Zipf is a good fit for the case of text, video and audio collections, if we use Jaccard similarity and the representations described earlier. For estimating the Zipf coefficient, POND uses peer sampling during the optimization step, as described earlier in this Section.

D. Minimizing the Network Cost of POND

We are now ready to find the values for k and l that minimize the network cost. Recall from Section IV that POND should find each near duplicate with a probability at least pr_{min}. By reducing Equation 2 for k and solving for $pr_{\text{found}} = pr_{min}$ and $Sim(x, y) = minSim$ we find the value of k that will return each resource with similarity at least $minSim$, with a probability higher than pr_{min}. This value, denoted as k_0, is: $k_0 = \frac{\log\left(1-(1-pr_{min})^{1/l}\right)}{\log\left(0.5-\frac{1}{2\cdot minSim}\right)+\log(minSim)}$

Next, we show that k_0 is also the value for k that minimizes network cost.

Theorem 6: For given l, $minSim$ and pr_{min} the value of k that minimizes network cost is $k = \lfloor k_0 \rfloor$

Proof Sketch: From Equation 2 we see that probability pr_{found} increases monotonically if k decreases and all other parameters are fixed. Therefore, for all $k \leq k_0$, the value of pr_{found} will be higher than or equal to the probability pr_{min} required by the user. Regarding network cost, the value of k is orthogonal to maintenance cost, but affects query execution cost due to the false positives. The number of false positives monotonically decreases with k. Therefore, the number of false positives will be reduced by selecting the maximum k value for which $pr_{\text{found}} \geq pr_{min}$. This value is $k = k_0$. ■

So far, we have assumed that the optimal value for parameter l is given. We now show how to eliminate this assumption. Finding theoretically the value of l that minimizes the cost is complicated. However, we can easily compute the optimal l using the following algorithm. We start with $l = 1$, and find the optimal k according to Theorem 6. For these pairs of k and l we compute the expected cost, according to the analysis presented in Section V-B. We then increment l, and repeat the computations. For some value of l, denoted with l_{inc}, the expected cost will start to increase, i.e., the cost for $l = l_{inc}$ will be higher than the cost for $l = l_{inc} - 1$. The optimal value of l is $l_{inc} - 1$. This holds because the cost function is convex, and therefore has only one minimum, namely at $l = l_{inc} - 1$. Note that this algorithm is executed once per republishing period, at the peer which performs algorithm optimization. It is computationally inexpensive and requires no network resources; thus, it does not constitute a bottleneck.

VI. Experimental evaluation

The purpose of our experimental evaluation was threefold. First, to compare POND with an NDD algorithm which does not optimize the values of k and l for demonstrating the significance of optimizing k and l dynamically. Second, to examine the efficiency and effectiveness of POND on different real-world collections and system configurations. Third, to evaluate the proposed extension for video linkage.

A. Datasets and Evaluation Setup

POND was evaluated on three large, real-world collections. As a text collection we have chosen REUTERS Corpus Volume I (RCV1) [13], a publicly available standard dataset in IR consisting of 802,253 newswire articles preprocessed using stemming and stopword filtering. It contains many near duplicates, which we used as queries for our evaluation. We computed a ground truth by comparing all articles pair-wise, and detecting all near duplicates. For videos, we constructed a collection by crawling a sample of Youtube videos with Tubekit [14][1]. The collection contains 22,455 videos with a total volume of 144 Gbytes. We did not consider existing publicly available video benchmarks such as the TRECVID collection because all of them just contain a relative small number of videos (less than 200), and therefore cannot be used for simulating P2P networks of reasonable sizes. For audio, we extracted the audio tracks of the described Youtube collection as MP3s, amounting to a total size of 82 Gbytes. The size of all three datasets sum up to 227 Gbytes.

For the text collection, we simulated a P2P network of 100,000 peers, structured over a Chord DHT [15]. All articles were distributed uniformly among the participating peers. For the video and audio collections which had less resources, we simulated a network of 1000 peers, distributing the resources uniformly among peers as well.

For constructing the query set we compared pair-wise all resources, and identified all pairs of resources with similarity higher than a threshold $minSim$ for $minSim \in \{0.8, 0.9\}$. Some of these pairs were in fact *exact* duplicates, e.g., the same video with a different file name. POND detected all exact duplicates, since these were always producing identical labels. Since these resources could also be detected with traditional hashing techniques, they were filtered out of the query set for the experiments reported here.

For each configuration, we measured the total network cost, i.e., the number of messages required for both maintenance and query execution. The quality for query execution was computed using the standard *recall measure*, i.e., the percentage of detected near duplicates. There was no need to measure precision separately, since this is always 1 due to

[1]We would like to thank the author Chirag Shah for assisting us with the crawling task.

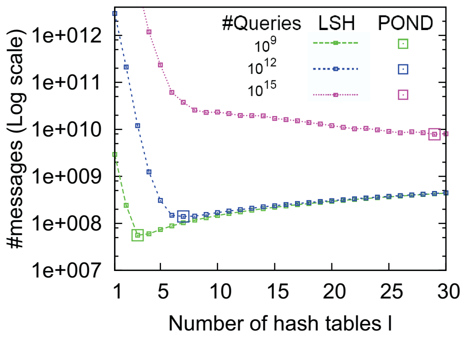

Fig. 3. Cost versus number of hash tables l, for $minSim = 0.9$

the way POND collects all near duplicates. As explained in Section IV, resources which are not near duplicates of the query are efficiently filtered out and are neither transmitted over the network to the query initiator, nor presented to the user as a result of the query.

B. Comparison with LSH

The advantage of POND compared to previous LSH algorithms is that it dynamically optimizes the values of k and l so that the total network cost is minimized. To find out how significant this optimization is for P2P networks, we compared POND with a P2P implementation of the original LSH [8], in which k and l are chosen manually by the user. In particular, we compared POND with the P2P algorithm $LSH(K)$ in [1]. Since the procedure of choosing l and k values is the only difference between the original LSH implementation and that of POND, the difference observed in the efficiency of the two algorithms can be directly attributed to the optimization performed by POND.

For this experiment, we indexed the collections in a network of 100,000 peers, and varied the rate of queries per republishing period between 10^6 and 10^{15}. Since original LSH does not offer an approach to select l and k, we tested it for all possible combinations of l and k that satisfied $pr_{min} = 0.9$. For each configuration we measured the total network cost, i.e., the aggregate cost for maintenance and query execution. The observed costs are shown in Fig. 3. Please note that Y axis uses logarithmic scale. The results correspond to the experiment with the RCV1 collection, which is the largest in terms of number of resources – the results with the video and audio collections were very similar, and are omitted. For clarity, the figure includes only the most efficient configuration for each l value.[2] For comparison, the figure also includes the cost incurred from POND for the same pr_{min} and query rates. As POND selects exactly one k and l combination for each query rate, its cost is presented as a single point.

We observe that the cost of original LSH depends heavily on the value of l. On the one hand, setting l too low leads to a very low k, and this increases the number of false positives by several orders of magnitude. On the other hand, setting l too high imposes the unnecessary overhead of maintaining more hash tables. The difference between the optimal and the incurred cost for each setting can be several orders of magnitude. More importantly, we see that there is no universally optimal value for l and k, but the optimal values depend on the system properties, which may even vary with time. In contrast, the default cost of POND for each setting is always almost equal to the global minimum cost of the original LSH; the extra network cost induced by the POND optimization step is negligible. This means that POND always finds the best configuration for l and k, and minimizes the cost.

We also repeated the experiment with the two multimedia collections, as well as with different network sizes. The experimental results were similar to the presented ones, additionally confirming the importance of optimizing the values of l and k for minimizing the network cost. In conclusion, by optimizing the values of l and k for the given configuration, POND can reduce the total network cost by several orders of magnitude.

C. Effectiveness for near duplicate detection

Effectiveness for NDD is measured with recall (precision is always 1, as explained in Section VI-A). As ground truth, we use all the true near duplicates, which are detected with a pair-wise comparison of all resources. Figure 4 presents recall for different values of pr_{min}, and for $minSim \in \{0.8, 0.9\}$. As expected, recall increases if pr_{min} is set to higher values. In particular, at higher pr_{min} values, the optimization step increases the number of hash tables or reduces the hash functions, in order to satisfy the higher probabilistic requirements. This behavior is consistent for the three different resource types.

We also see that even for very low pr_{min} values, recall values remain within acceptable levels. For example, for the RCV1 corpus, recall is always higher than 0.8, even for $pr_{min} = 0.5$. The reason for this is that POND computes the probabilistic guarantees considering the minimum similarity value $minSim$, whereas most of the near duplicates have similarity higher than $minSim$. Therefore, the probability that a near duplicate is found is higher than the obtained probabilistic guarantees for most of the near duplicates. For the same reason, recall increases slightly if $minSim$ is 0.8, compared to $minSim = 0.9$.

D. Network cost

Figure 5 shows network costs for different values of pr_{min}, with $minSim$ set to 0.9. All plots present the cost split into two types: (a) *maintenance cost*, which is the average number of messages required by POND for indexing a single resource in the network, and, (b) *query cost*, corresponding to the average number of messages required for detecting and retrieving all near duplicates of a query, and for filtering out all non near-duplicates.

We observe that both indexing cost and query execution cost remain within reasonable limits. For example, for the RCV1 experiment with $pr_{min} = 0.8$, maintenance requires less than 40 messages per article, whereas querying requires

[2]With respect to the original LSH, for a given pair of l and k values that satisfy the probabilistic guarantees, all other configurations with the same l but with $k' < k$ also satisfy the probabilistic guarantees, albeit with higher costs.

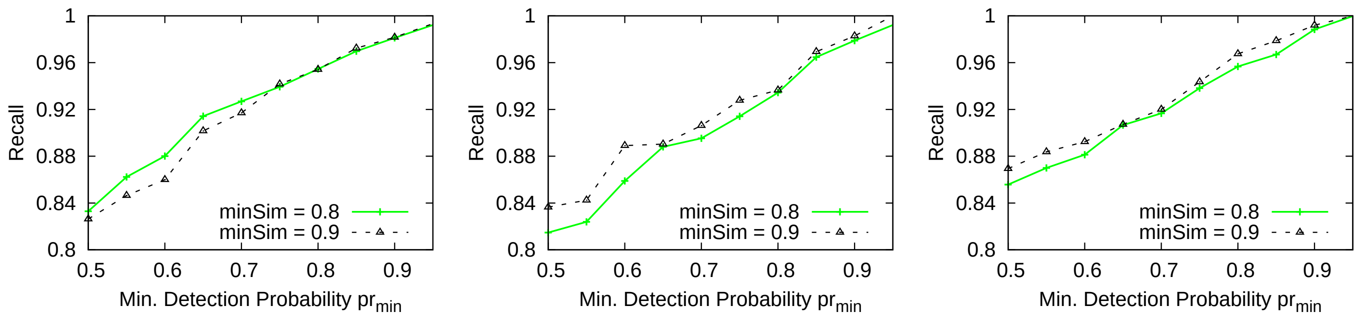

Fig. 4. Recall versus minimum detection probability pr_{min}: (a) RCV1 collection, (b) Video collection, (c) Audio collection

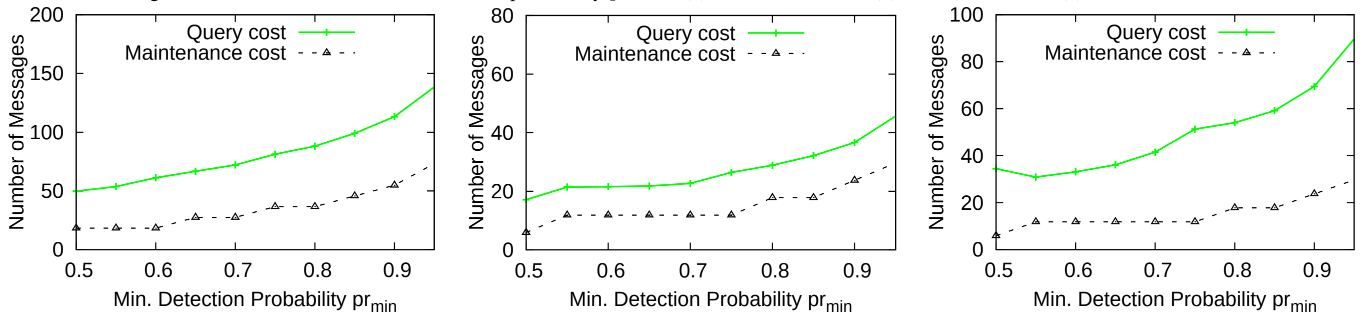

Fig. 5. Cost versus minimum detection probability pr_{min} (a) RCV1 collection, (b) Video collection, (c) Audio collection

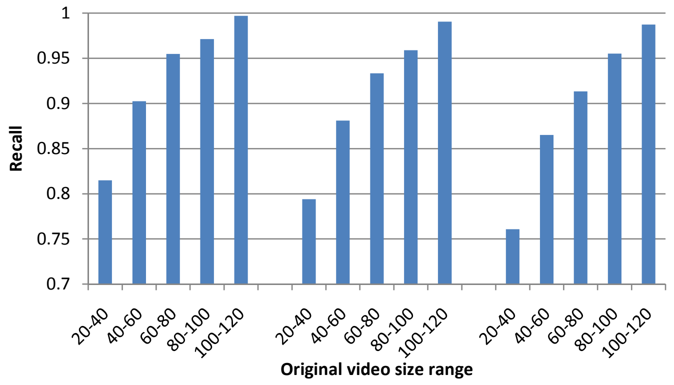

Fig. 6. Video Linkage: Recall versus average file size for videos broken to 2, 3 and 4 parts

less than 80. Note that for the aforementioned configuration, recall is already above 0.95. Furthermore, as expected, increasing pr_{min} causes higher network cost, because POND increases the number of hash tables l to satisfy the required probabilistic guarantees. Nevertheless, even for $pr_{min} = 0.95$ which provides a recall of almost 1, maintenance cost does not surpass 70 messages per resource. These observations are valid also for the audio and video collections.

We also see that query execution cost is always higher than maintenance cost. As described in detail in Section V-B, maintenance cost involves performing l DHT lookups, while query execution cost additionally requires contacting all candidate peers and collecting all the near duplicates. Therefore, maintenance cost per resource is expected to be less than query execution cost.

Finally, it is interesting to compare the costs incurred for different types of resources. We observe that cost related to text documents is slightly higher than the respective cost for video and audio resources. This difference is mostly attributed to the difference of network sizes in the corresponding experimental setups as it affects the cost of DHT lookups. Since POND uses DHT lookups both for maintenance and query execution, the respective costs are also affected. The increase is only logarithmic to the network size. We also observe that querying costs for audios and videos differ slightly, even though the audio resources are generated from the videos. This happens because there are more near duplicates in the audio collection, compared to the video collection. The most typical case in which two audios were near duplicates while their corresponding videos were not, was songs accompanied by slide-shows, i.e. two videos having the exact same song as audio but showing completely different slide-shows.

E. Video Linkage

We also evaluated POND for the video linkage problem. For generating a query set $\mathcal{Q}$, we selected a subset of videos with file sizes uniformly distributed between 20 and 120 Mbytes, and split them to 2, 3, or 4 equal-sized parts. We then used all partial videos $q \in \mathcal{Q}$ as queries for POND. A query was considered successful if it returned the original video from which it was created. Similar to the previous experiments, precision was always 1 due to the way POND collects the near duplicates (see Section VI-A).

Figure 6 plots the average recall for different video file sizes. The algorithm is initialized with $pr_{min} = 0.9$ and $minSim = 0.9$, and runs on a network of 1000 peers. We observe that recall increases significantly with the size of the original video. For example, recall for the smaller files (20 – 40 Mbytes) is around 0.75, while recall for larger files (100 –

120 Mbytes) is almost 1. This is expected: since each query segment is handled as an individual resource, the probability that a query succeeds increases exponentially with the number of overlapping segments between the query and the full video. The number of common segments increases with the full video size, and with the query size. For the same reason, recall is below the expected value of 0.9 for very small files, and also decreases slightly with the number of splits. However, for large videos, like the ones that often occur on the Internet, recall is practically 1.

With respect to network cost, the cost for video linkage is typically higher than the cost of NDD because each video segment is indexed and looked-up individually. In our experiments, maintenance cost for video linkage was at most 110 messages per video, which is easily affordable. This cost is distributed uniformly among all participating peers, as it is mostly composed of DHT lookups. Additional cost optimizations can be achieved by reducing the number of segments in each video, i.e., by increasing the distance threshold for splitting a video to segments. Including this threshold in the optimization analysis is part of our future work.

In conclusion, POND efficiently and effectively addresses the video linkage problem. Especially with respect to large videos, like the majority of the videos on the internet, it offers a recall very close to one.

VII. Conclusions and Future Work

In this paper we presented POND, a novel algorithm for near duplicate detection whose key innovation lies in the deployment of parameter optimization to minimize network usage. We derived probabilistic guarantees for the success of POND to answer NDD queries, and showed how it automatically configures the core parameters of the underlying indexing method. Furthermore, we extended POND to address the video linkage problem. A large-scale experimental evaluation using real world datasets of more than 200 Gbytes demonstrated the importance of optimizing the LSH parameters for reducing the cost of NDD in P2P networks. The results confirm that POND successfully optimizes the LSH parameters, reduces the network cost to the theoretical minimum, and satisfies the required probabilistic guarantees. The incurred network cost is easily affordable by the participating peers, even for huge networks and collections, making the algorithm suitable for integration in any mainstream DHT-based file-sharing system.

Regarding future work, we aim to repeat the theoretical analysis of POND for other P2P network configurations [1], [9]. These systems currently address NDD as a special case of the KNN problem, and therefore POND cannot be applied directly to optimize the network cost. Specializing the functionality of these systems to only NDD queries with fixed distance thresholds would enable the application of our optimization method. Furthermore, we will investigate how the performance of POND is affected by the similarity measures and the resource representations of multimedia files, allowing for a more directed choice of similarity functions to help further reducing network costs in P2P systems. Finally, we will explore new application scenarios of NDD in multimedia mining and retrieval over existing P2P file sharing networks, such as Limewire. Specifically, we believe that the proposed technique has direct applications to metadata-based search, where annotations (such as tags or descriptions) provided by different users for near duplicate content can be combined to build more comprehensive indices.

Acknowledgements

This work is partially supported by the FP7 EU Project GLOCAL (contract no. 248984)

References

[1] M. Bawa, T. Condie, and P. Ganesan. LSH forest: self-tuning indexes for similarity search. In *WWW*, pages 651–660, 2005.

[2] A. Z. Broder, M. Charikar, A. M. Frieze, and M. Mitzenmacher. Min-wise independent permutations (extended abstract). In *STOC*, pages 327–336, 1998.

[3] M. Datar, N. Immorlica, P. Indyk, and V. S. Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. In *Proceedings of the twentieth annual symposium on Computational geometry (SCG)*, 2004.

[4] W. Dong, Z. Wang, W. Josephson, M. Charikar, and K. Li. Modeling LSH for performance tuning. In *CIKM*, pages 669–678. ACM, 2008.

[5] S. El-Ansary, L. O. Alima, P. Brand, and S. Haridi. Efficient broadcast in structured P2P networks. In *IPTPS*, pages 304–314, 2003.

[6] F. Falchi, C. Gennaro, F. Rabitti, and P. Zezula. A distributed incremental nearest neighbor algorithm. In *INFOSCALE*, page 82, 2007.

[7] A. J. Ganesh, A.-M. Kermarrec, E. L. Merrer, and L. Massoulié. Peer counting and sampling in overlay networks based on random walks. *Distributed Computing*, 20(4):267–278, 2007.

[8] A. Gionis, P. Indyk, and R. Motwani. Similarity search in high dimensions via hashing. In *VLDB*, pages 518–529, 1999.

[9] P. Haghani, S. Michel, and K. Aberer. Distributed similarity search in high dimensions using locality sensitive hashing. In *EDBT*, 2009.

[10] Y. Ke, R. Sukthankar, L. Huston, Y. Ke, and R. Sukthankar. Efficient near-duplicate detection and sub-image retrieval. In *ACM Multimedia*, 2004.

[11] H.-s. Kim, J. Lee, H. Liu, and D. Lee. Video linkage: group based copied video detection. In *CIVR*, 2008.

[12] V. A. Larsen. Combining audio fingerprints. Technical report, Norwegian University of Science and Technology, 2008. Available at http://daim.idi.ntnu.no/.

[13] D. D. Lewis, Y. Yang, T. G. Rose, and F. Li. RCV1: A new benchmark collection for text categorization research. *Journal of Machine Learning Research*, 5:361–397, 2004.

[14] C. Shah. Tubekit: a query-based youtube crawling toolkit. In *Joint Conference of Digital Libraries (JCDL)*, page 433, 2008.

[15] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. *SIGCOMM Computer Communication Review*, 31(4):149–160, 2001.

[16] X. Wu, A. G. Hauptmann, and C.-W. Ngo. Practical elimination of near-duplicates from web video search. In *MULTIMEDIA '07*, pages 218–227, 2007.

[17] X. Wu, C.-W. Ngo, and Q. Li. Threading and autodocumenting news videos: a promising solution to rapidly browse news topics. *Signal Processing Magazine, IEEE*, 23(2):59–68, March 2006.

[18] C. Xiao, W. Wang, X. Lin, and J. X. Yu. Efficient similarity joins for near duplicate detection. In *WWW*, New York, NY, USA, 2008. ACM.

[19] Y. Yan, B. C. Ooi, and A. Zhou. Continuous content-based copy detection over streaming videos. In *ICDE*, pages 853–862, 2008.

[20] C. Yang. Peer-to-peer architecture for content-based music retrieval on acoustic data. In *WWW*, pages 376–383, 2003.

[21] H. Zhang, A. Kankanhalli, and S. W. Smoliar. Automatic partitioning of full-motion video. *Multimedia Systems*, 1(1):10–28, 1993.

[22] X. Zhou, L. Chen, A. Bouguettaya, N. Xiao, and J. A. Taylor. An efficient near-duplicate video shot detection method using shot-based interest points. *IEEE Transactions on Multimedia*, 11(5):879–891, 2009.

[23] B. Zhu, K. Li, and H. Patterson. Avoiding the disk bottleneck in the data domain deduplication file system. In *6th USENIX Conference on File and Storage Technologies (FAST)*, pages 1–14, 2008.