MediaScope: Selective On-Demand Media Retrieval from Mobile Devices

Paper #53

ABSTRACT

Motivated by an availability gap for visual media, where images and videos are uploaded from mobile devices well after they are generated, we explore the *selective, timely retrieval* of media content from a collection of mobile devices. We envision this capability being driven by *similarity-based queries* posed to a cloud search front-end, which in turn dynamically retrieves media objects from mobile devices that best match the respective queries within a given time limit. Building upon a crowd-sensing framework, we have designed and implemented a system called MediaScope that provides this capability. MediaScope is an extensible framework that supports nearest-neighbor and other geometric queries on the feature space (e.g., clusters, spanners), and contains novel retrieval algorithms that attempt to maximize the retrieval of relevant information. From experiments on a prototype, MediaScope is shown to achieve near-optimal query completeness and low to moderate overhead on mobile devices.

1. INTRODUCTION

Cameras on mobile devices have given rise to significant *sharing* of media sensor data (photos and videos). Users upload visual media to online social networks like Facebook [2], as well as to dedicated sharing sites like Flickr [4] and Instagram [3]. However, these uploads are often not *immediate*. Camera sensors on mobile devices have been increasing in both image and video resolution far faster than cellular network capacity. More important, in response to growing demand and consequent contention for wireless spectrum, cellular data providers have imposed data usage limits, which dis-incentivize immediate photo uploading and create an *availability gap* (the time between when a photo or image is taken and when it is uploaded). This availability gap can be on the order of several days (Section 2).

The availability gap for visual media represents missed opportunities. Consider the following scenario: users at a mall or some other location take pictures and video of some event (e.g., an accident or altercation). An investigative team wants visual evidence of the event, and could have searched or browsed images on, say, of photo sharing service such as Flickr, and may have been able to retrieve evidence in a timely fashion if the images had been immediately uploaded.

To bridge this availability gap, and to enable this and other missed opportunities (Section 2), we consider a novel (Section 5) capability for selective and timely on-demand retrieval of images from mobile devices. Specifically, we develop a system called MediaScope that permits concurrent timely geometric queries in feature space on media data that may be distributed across several mobile devices. MediaScope is an extensible framework that supports different kinds of queries (Section 3). MediaScope queries permit nearest-neighbor searches on image feature spaces, *spanners* that the retrieve samples of images that span the feature space, or *cluster representatives* that are samples of clusters in the feature space.

Queries may have a timeliness bound associated with them, and an important challenge in the design of MediaScope is to retrieve query results, in the presence of *concurrent queries*, while respecting the timeliness constraints. MediaScope addresses this challenge using an approach that trades off query completeness while meeting timeliness requirements. It incorporates a novel credit-assignment scheme that is used to differentially weight queries as well as differentiate query results by their "importance". A novel credit and timeliness-aware scheduling algorithm that also adapts to wireless bandwidth variability ensures that query completeness (measured by the total credit of all the returned results of the query) is optimized. A second important challenge is to enable accurate yet computationally-feasible feature extraction. MediaScope addresses this challenge by finding sweet spots in the trade-off between accuracy and computational cost, for extracting features from images and frames from videos.

An evaluation MediaScope on a complete prototype (Section 4), shows that MediaScope achieves upwards of 75% query completeness even in adversarial settings. For the query mixes we have experimented with, this completeness rate is near-optimal; an omniscient scheduler that is aware of future query arrivals does not outperform MediaScope. Furthermore, MediaScope's performance is significantly different from other scheduling algorithms that lack one of its features, namely timeliness-awareness, credit-awareness, and

adaptivity to varying bandwidth; this justifies our choice of algorithm. Finally, we quantify the overheads associated with different components in the MediaScope prototype, and find that most overheads are moderate, suggesting that timeliness bounds within 10s can be are achievable.

2. MOTIVATION AND CHALLENGES

In this section, we first motivate the need for on-demand image retrieval, then describe our approach and illustrate the challenges facing on-demand image retrieval.

Motivation. With the increasing penetration of mobile devices with high-resolution imaging sensors, point-and-shoot cameras and camcorders are increasingly being replaced by mobile devices for taking photos and videos. This trend is being accelerated by the increase in resolution of image sensors to the point where mobile devices have image resolutions comparable to cameras.

The availability of high resolution image sensors has prompted users to more pervasively share images and videos. In addition to giving birth to services like Instagram, it has prompted many image and video sharing sites to develop a business strategy developed on mobile devices. Beyond sharing media (photos and videos) with one's social network, this development has also been societally beneficial, e.g., in crime-fighting [1].

On the flip side, wireless spectrum is scarce and has not been able to keep up with increases in mobile device usage. As a result, cellular operators limit data usage on mobile devices; standard data plans come with fairly restrictive data usage budgets per month (on the order of 1-2 GB). Users are increasingly becoming aware of the implications of these limits and how media transmission can cause users to exceed their monthly data usage limits.

These conflicting trends will, we posit, lead to an *availability gap* for media. The availability gap for a media item (an image or a video) is defined as the time between which the item is taken and when it is shared (uploaded to a sharing site). We believe that users will be increasingly reluctant to use cellular networks to share media, preferring instead to wait for available WiFi. Indeed, this availability gap already exists. Figure 1 shows the CDF of the availability gaps for 50 photos each from 40 Flickr users who were featured in Flickr's "Calendar view of this month". This shows that more than 50% of the photos on Flickr have an availability gap of greater than 10 days!

We conjecture that this availability gap will persist with mobile devices: existing data plan usage limits ensure that users treats these devices as similar to traditional cameras or camcorders from the perspective of video and photo upload (i.e., as a device with no network connectivity). Furthermore, mobile device storage has been increasing to the point where multiple photos and videos can be stored; a 64GB iPad can hold 10,000 photos that can take several months to upload with a 2GB/month data plan.

This availability gap represents a missed opportunity for societal or commercial uses. Consider these examples:

- Consider a robbery in a mall in an area uncovered by security cameras. The mall's security staff would like to be able to access and retrieve images from mobile devices of users who happen to be in the mall on that day in order to be able to establish the identity of the thief.
- A sportswriter is writing a report on a sporting event and would like to be able to include a perfect picture of a play (e.g., a catch or a dunk). The newspaper's staff photographer happened to have been obscured when the play happened, so the sportswriter would like to be able to retrieve images from mobile devices of users who happened to be attending the event.
- A talent scout [5] is looking for the next musical sensation and would like to be able to retrieve clips of budding stars. The ability to retrieve video clips, of people singing, taken at parties and other events, can accelerate this search.

The focus of this paper is the exploration of a capability for bridging the availability gap by enabling media retrieval in a manner suggested by the above examples.

Approach. To bridge the availability gap, we explore on-demand retrieval of images from a collection of mobile devices. These devices belong to users who have chosen to *participate* and provide images on demand. In return, participating users may be incentivized by explicit micropayments; we do not discuss incentives and privacy issues in this paper, but note that our approach is an instance of crowd-sensing, and recent work has explored these issues in the context of crowd sensing [25]. In what follows, we use the term *participating device* to mean a mobile device whose user has chosen to participate in image retrieval.

Our approach is inspired by *image search* techniques that support similarity searches on image feature space. There is a large body of literature that seeks support *content-based image retrieval* in order to define appropriate features that characterize images: ImgSeek[19], CEDD [10] (Color and Edge Directivity Descriptor), FCTH [11] (Fuzzy Color and Texture Histogram), Auto Color Correlogram [18], and JCD [12] (Joint Composite Descriptor). Generally, these algorithms are based on 2 features: image color and texture desciption. Taking CEDD as an example, for texture space, CEDD sub-divides an image into blocks and for each image block, sub-divides it into 4 sub-blocks, calculates the average gray level of each sub-block, then computes the directional area (vertical, horizontal, 45-degrees, 135-degrees and non-directinal) with the sub-block parameters for this image block; thus, an image is divided to 6 regions by texture unit. For color space, it projects the color space into HSV (Hue, Saturation, Value) channels, then divides each channel into several preset areas using coordinate logic filters (CLF), so that the color space is divided to 24 sub-regions. A histogram is drawn on these parameters, so that 24x6 = 144 coefficients are output as the CEDD feature vector. Finally, the image

processing community has experimented with a wide variety of measures of similarity. Of these, we pick a popular measure [10, 11, 23], the Tanimoto distance [26], which satisfies the properties for a metric space [22].

Since CEDD is popularly used and widely accepted, we have developed our system (Section 3) using this algorithm. From our perspective, this algorithm has one important property: for a single image, CEDD's feature vectors consist of 144 coefficients, requiring significantly smaller storage and communication overhead than the original image which may be several megabytes in size. CEDD is defined for images; as we describe later, we are also able to derive features for video. More generally, our approach is agnostic to the specific choice of features and similarity definition; other feature extraction algorithms can be used, so long as the features are compact relative to image sizes.

On top of this image similarity search primitive, we explore a query interface that supports several queries:

Top-K Given an image, this query outputs the K most similar images among all images from all participating devices.

Spanners This query returns a collection of images whose features span the feature space of all images from all participating devices.

Clusters This query returns representatives from natural clusters in the feature space and can effectively identify the most common "topics" among images from participating mobile devices.

Our approach can be extended to support other kinds of queries (e.g., enclosing *hulls*), as described later.

Our queries can be *qualified* by several *attributes*. Attributes like *location* and *time* constrain the set of objects that are considered in computing the query result; the location attribute constrains media objects to those taken in the vicinity of a certain location and the time attribute specifies when the corresponding photo or video was taken. Users may also specify a *freshness* attribute, which constrains the age of media objects selected to compute the query result.

The last, but perhaps the most interesting attribute, is *timeliness*. Timeliness is a property of the query result, and specifies a time bound within which to return the result(s) of a query. The timeliness attribute is motivated by the surveillance example discussed above; the security team might want results within a bounded time to take follow-up action.

In summary, our approach bridges the availability gap by extracting relevant photos and images dynamically from participating devices. The approach hinges on the observation that feature space similarity can be used to determine relevant media objects, and that image features are an extremely compact representation of the contents of an image. However, it is well-known that content based information retrieval exhibits a *semantic gap* [29]: feature-based similarity matching is oblivious to the semantic structures within an image, so the matching may not be perfect. In these cases, we rely on additional filtering by human intelligence (e.g., in our examples, the security officer, or the reporter).

Challenges. Our approach faces several challenges. The first of these is *feature extraction*: it turns out that feature extraction algorithms for large images encounter memory limits even on high-end modern smartphones. Equally challenging is feature extraction for video, since the frame rate for video can overwhelm many feature extraction algorithms.

The more central challenge in our work is the design of the system that *satisfies the timeliness constraints multiple concurrent queries*. In general, this is a hard problem, primarily because of the bandwidth limitations of wireless mobile devices; the aggregate query result may need a throughput that may overwhelm the available bandwidth. There are two approaches to solve this problem. The first is admission control, whereby we restrict the number of concurrent queries such that the timeliness constraints can always be met. We did not consider this solution because of the variability and unpredictability of wireless bandwidth availability. The second approach is to deliver maximal *information* within the given timeliness bound, while adapting to variability in available bandwidth. Our work chooses the second approach, in the context of which there is an interesting challenge: what does it mean to deliver maximal information?

In the next section, we describe the design of a system called *MediaScope* that addresses these challenges.

3. MEDIASCOPE

MediaScope is a system that supports timely similarity-based queries on media objects stored on mobile devices. We begin by describing the MediaScope architecture and then discuss the design and implementation of each component.

3.1 Architecture and Overview

Mediascope is conceptually partitioned across a cloud component called MSCloud, and another component called MSMobile that runs on mobile devices. This partitioned design leverages the computation and storage power of clouds to support geometric queries on the feature space; mobile devices provide sensing and storage for media objects.

These components interact as follows (Figure 2). In the background, whenever participants take photos or videos, the *Feature Extractor* component of MSMobile continuously extracts image and video features and uploads them to the *MSCloudDB*. Users (e.g., a security officer or a reporter) pose queries to MSCloud using a standard web interface, possibly on a mobile device. These queries are processed by the *MSCloudQ* query processing engine, which uses the features stored in the MSCloudDB to compute the query results. The results of the queries identify the media objects that need to be retrieved from individual mobile devices. In some cases, a media object may already have been retrieved as a result of an earlier query; query results are also *cached* in MSCloudDB in order to optimize retrieval. MSCloudQ coordinates with an *Object Uploader* component on MSMo-

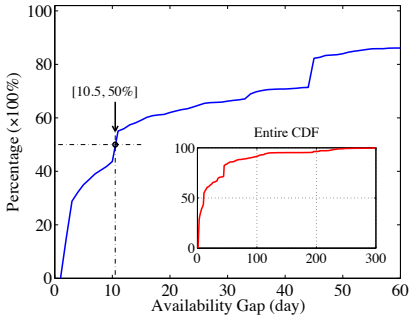


Figure 1—CDF of Flickr Photo Availablility Gap (days)

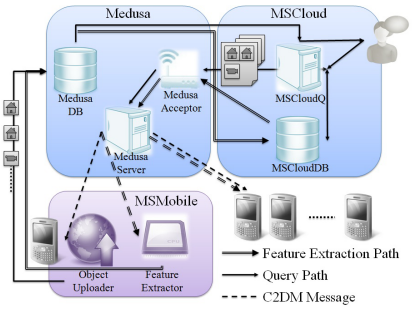


Figure 2—System Architecture Work Flow

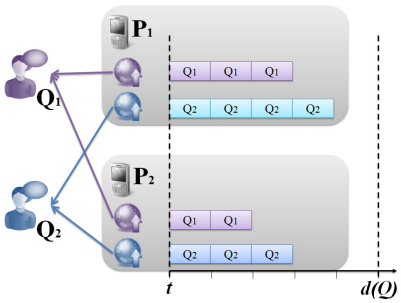


Figure 3—Illustration of Concurrent Queries

bile in order to retrieve query results.

MediaScope uses a publicly available crowd sensing platform called Medusa [25]. Medusa was originally designed to permit human users to pose crowd-sensing tasks. MediaScope's retrieval of features and media objects from mobile devices leverages Medusa's support for "sensing" stored information on these devices. To enable programmed interaction between MSCloud and Medusa, and to support MediaScope's timeliness requirements, we made several modifications to the Medusa platform (discussed later).

MediaScope thus provides a high-level abstraction (queries on media objects) that hides many of the details of object retrieval from users. In the following subsections, we describe the two most challenging aspects of MediaScope design: *support for concurrent queries*, a functionality distributed between the MSCloudQ and the Object Uploader; and *feature extraction*. We conclude with a brief description of other design and implementation issues.

3.2 Design: Concurrent Queries

The most challenging component of MediaScope is support for concurrent queries. In MediaScope, the result of a query is a list of media objects to be retrieved from a subset of the participating phones. Recall that each query has a timeliness constraint. In the presence of concurrent queries, MediaScope may need to upload all media objects before their timeliness bound expires. In general, this may be difficult to achieve because wireless bandwidth is a bottleneck.

To illustrate this, consider the example of two concurrent queries Q_1 and Q_2 that arrive at the same time for media objects distributed across two phones P_1 and P_2 in Figure 3. Also, assume that both queries have a timeliness bound of 5 seconds, each object can upload 1 object per second, and all objects are of the same size. If Q_1 needs to retrieve 3 objects from P_1 and 2 objects from P_2, while Q_2 needs to retrieve 4 objects from P_1 and 3 from P_2. Under these circumstances, it is not possible to satisfy the timeliness requirements of one of the two queries. In practice, the problem is much harder because there may be more than two concurrent queries, many more participating devices, queries can arrive at different times, media objects may have different sizes, and wireless available bandwidth can vary dynamically. Especially because of the last reason, a solution that controls the admission of queries may be insufficient (it cannot guarantee that all timeliness constraints are met, or may severely underutilize the available bandwidth).

MediaScope uses a different approach, *trading off query completeness for timeliness*. In MediaScope, not all query results may be uploaded within the timeliness bound, but the challenge is to upload the most relevant queries so as to maximize the amount of *information* retrieved. In doing this, there are two challenges: how to differentiate between queries, and how to prioritize media items for the retrieval in order to maximize the information retrieved.

MediaScope addresses these two challenges using a *credit assignment* mechanism. Each query is assigned, by MediaScope, a number of credits. The credits assigned to a query reflect the importance of that query and result in proportionally more information being uploaded for that query (and therefore the proportional completeness of the query result). The specific credit assignment mechanism for queries is beyond the scope of this paper, but MediaScope may use monetary incentives (e.g., users who pay more get more credits) or other approaches in order to assign credits to queries.

If a query is assigned n credits, it divides up these credits among its results (media objects) in a way that reflects the importance of each object to the query. The key intuition here is that, for a given query, the importance of a result object to the query can be determined by the feature space geometry. For example, consider a query Q which attempts to retrieve the two nearest photos in feature space to a given photo c. If the resulting photos a and b are each 20 units and 80 units distant from c in feature space, and Q has been assigned 100 credits, a and b each receive 80 and 20 credits respectively (in inverse proportion to their distances to c).

MediaScope uses this intuition to define credit assignment to result objects. Once objects have been assigned credits, object uploading is prioritized by credit in order to maximize the total credit retrieved across all concurrent queries. In what follows, we first describe the queries that MediaScope supports and how credits are assigned for each query. We then describe MediaScope's credit-based object scheduling technique and discuss its optimality.

3.2.1 Queries and Credit Assignment

Our current instantiation of MediaScope supports three qualitatively different queries: nearest neighbor, clusters, and spanners. Below, we discuss the design of the query engine MSCloudQ and how credits are assigned to query results. Recall that for each query, users can specify time, location and freshness attributes: before performing each of the queries described below, MSCloudQ filters all the feature vectors stored in MSCloudDB to select feature vectors that match these attributes. In our description of the queries below, we assume that this filtering step has been applied.

k-Nearest Neighbors. For this query, the user supplies a *target* image and the server attempts to return the k nearest images (from photos or videos) in feature space to the target. The implementation of this query is straightforward: it is possible to build indexes to optimize the search for the K nearest neighbors, but our current implementation uses a brute force approach.

Credit assignment for this query attempts to capture the relative importance of the query results. Thus, the assignment of credits to each result is proportional to its similarity to the target image. For the i-th result, let s_i be the similarity measure to the target; we then assign credits to the i-th result proportional to $p_i = (1 - \frac{s_i}{\sum s_i})$.

K Clustering. The second class of queries supported by MSCloudQ is based on clustering in feature space. This query takes as input the number k as well as well as a *type* parameter which describes the expected result and can have one of two values:

Cluster Representative With this parameter, the result contains k images, one from each cluster. For each cluster, our algorithm selects that image as the representative whose distance is least to the centroid of the cluster. Intuitively, this query type identifies different "topics" among images taken by participating users.

Common Interest With this parameter, the result includes images from that cluster which contains objects belonging to the most number of users. Thus, if the i-th cluster contains images from u_i users, the query returns images from that cluster for which u_i is the largest. Intuitively, this query identifies the cluster that represents the maximal common interest between participating users. Within the selected cluster, the query returns one image for each participating user, selecting that image of the user that is closest to the centroid of the cluster.

These queries can be implemented by any standard algorithm for k-means clustering.

For the *cluster representative* type of query, we assign credits proportional to the size of the cluster. Thus, if the j-th cluster's size is c_j, the credit assigned to the image selected from cluster j is proportional to $\frac{c_j}{\sum c_j}$.

For the *common interest* type of query, we assign a credit to each selected image that is inversely proportional to the image's distance from the centroid of the cluster. The credit assignment is similar to k nearest neighbors above.

Spanner. The third, and qualitatively different query that MediaScope supports is based on spanning the feature space. The intuition behind the query is to return a collection of images which *span* the feature space. In computing the spanner, we assume that each user t contributes exactly s_t images, where s_t is derived from the query's timeliness bound and a nominal estimate of the average uploading rate from the corresponding mobile device. Our spanner maximizes the minimum dissimilarity between all pairs.

We now express this problem mathematically. Assume that K_n, the complete graph on n vertices (vertices represent images), has a vertex set V partitioned into C classes $V_1, \ldots, V_C$ (classes represent users). Let v_{i_t} denote vertex i in class V_t. Let $e_{i_t j_k}$ represent the edge connecting v_{i_t} with v_{j_k}. Assume edge $e_{i_t j_k}$ has weight $w_{i_t j_k}$ (where the weight represents the dissimilarity between objects i_t and j_k).

Assuming that exactly s_t vertices must be selected from V_t, select a set of vertices so that the minimum edge weight of the selected clique is maximized. This problem can be formulated as a mixed-integer program:

$$
\begin{aligned}
&\max z \\
&\text{s.t. } z \le w_{i_t j_k} y_{i_t j_k} && \forall i_t < j_k && (1)\\
&\quad y_{i_t j_k} \le x_{i_t} && \forall i_t < j_k && (2)\\
&\quad y_{i_t j_k} \le x_{j_k} && \forall i_t < j_k && (3)\\
&\quad x_{i_t} + x_{j_k} - y_{i_t j_k} \le 1 && \forall i_t < j_k && (4)\\
&\quad \sum_{i_t \in V_t} x_{i_t} = s_t && \forall t = 1, \ldots, C && (5)\\
&\quad x_{i_t} \in \{0,1\} && \forall i_t\\
&\quad y_{i_t j_k} \in \{0,1\} && \forall i_t \ne j_k
\end{aligned}
$$

In this mixed-integer program, variable x_{i_t} is used as the indicator variable for selecting vertex v_{i_t} for the clique. Similarly, variable $y_{i_t j_k}$ is used as the indicator variable for selecting edge $e_{i_t j_k}$ for the clique. Variable z is used to achieve the $\min_{i_t < j_k} w_{i_t j_k} y_{i_t j_k}$. Inequalities 2 and 3 ensure that edge $e_{i_t j_k}$ is not selected if either vertex i_t or j_k is not selected. Inequality 4 guarantees that $y_{i_t j_k}$ is selected if both both vertices i_t and j_k are selected. Inequality 5 ensures that the number of vertices selected from class t is s_t.

The above problem is NP-hard so we use a $O(|V|^2 log(|V|))$ heuristic (Algorithm 1). The idea behind this heuristic is to select the set of vertices greedily i.e., add vertices whose minimum weighted edge to the set selected thus far is maximum. We deal with the issue of which vertex should be selected first by trying all possible vertices as being the first vertex in the set and taking the maximal such set.

Algorithm 1 : MAXMIN HEURISTIC

Define a list l for storing best vertex set and a variable max_min for minimum weighted edge
$l \leftarrow []$, $max\_min \leftarrow 0$
for all $i \in \{1, \ldots, V\}$ **do**
  $min = \infty$
  Define a temporary list l_t and $l_t \leftarrow i$
  while new item added to l_t **do**
    for $j \in \{1, \ldots, V\}$ and $j \notin L$ **do**
      $d(j) \leftarrow \min_{o \in l_t} similarity\_dist(o, j)$
    if $\exists$ qualified vertice v **then**
      $l_t.add(\{v | \max d(v)\})$
      $temp\_min \leftarrow d(\{v | \max d(v)\})$
    if $temp\_min < min$ **then**
      $min = temp\_min$
  if $min > max\_min$ **then**
    $max\_min = min$
    $l = l_t$

OUTPUT: l and max_min

For this query, intuitively, credit assignment should give more importance to dissimilar images. For the i-th query result, we compute d_i, the average distance from the i-th image to all other images. The credit assigned to this image is proportional to $\frac{d_i}{\sum d_i}$.

Extensibility of MSCloudQ. These are, of course, not the only kinds of geometric queries that can be supported. Developers wishing to extend MSCloudQ by adding new queries can do so quite easily by: (a) defining the query syntax and semantics, (b) implementing the query algorithm, and (c) specifying a proportional credit assignment based on the semantics of the query.

3.2.2 Credit-based Scheduling

In general, users can pose concurrent queries to MSCloudQ. Queries may arrive at different times and may overlap to different extents (we say one query overlaps with another when one arrives while the other's results are being retrieved). Furthermore, different queries may have different timeliness constraints, may retrieve different numbers of objects (e.g., for different values of k, or different sizes of spanners), and the retrieved media objects may be of different sizes (images with different resolutions). In these cases, MSCloudQ needs an algorithm that schedules the retrieval of different objects subject to some desired goal.

In MediaScope, this goal is to maximize the total completeness of queries, defined as the sum of the credits of all the uploaded images. To achieve this, recall that MSCloudQ assigns a credit budget to each query based on the importance of that query; then, using the proportions defined above, it assigns credit values to each query result.

To mathematically define the completeness goal, we first introduce some notation. Let Q_i denote the set of media objects that form the result of the i-th query, and let that query's timeliness constraint be $d(Q_i)$. Let $g(o)$ be an indicator variable that denotes whether a media object o is retrieved before $d(Q_i)$. Then, for the i-th query, the total credit for all uploaded media objects is given by:

$$g(Q_i) = \sum_{o \in Q_i} g(o) \cdot c(o)$$

Thus, given a series of concurrent queries $\mathbb{Q}$, the total number of credits retrieved is given by:

$$c(\mathbb{Q}) = \sum_{Q \in \mathbb{Q}} \sum_{o \in Q} g(o) \cdot c(o)$$

Maximizing this quantity is the objective of MediaScope's retrieval scheduling algorithm.

It turns out that it is possible to decompose this objective into a per-device *credit maximization scheduling* algorithm. To see why this is so, let $\mathbb{P}$ denote the set of participating devices, and the k-th device be denoted by p_k. Then, the above credit sum can be written, for concurrent queries $\mathbb{Q}$:

$$\begin{aligned} c(\mathbb{Q}) &= \sum_{Q \in \mathbb{Q}} \sum_{o \in Q} g(o) \cdot c(o) \\ &= \sum_{Q \in \mathbb{Q}} \sum_{P \in \mathbb{P}} \sum_{o \in P \cap Q} g(o) \cdot c(o) \\ &= \sum_{P \in \mathbb{P}} \sum_{o \in P} g(o) \cdot c(o) \end{aligned}$$

This equality shows that, in order to maximize the total credits retrieved across a set of concurrent queries $c(\mathbb{Q})$, it suffices to maximize the total credits uploaded by each participating device: $\sum_{P \in \mathbb{P}} c(P)$. This is true under the following two assumptions: (a) if two different queries retrieve the same object from P_k, that object is uploaded only once if it is uploaded at all, and (b) the credit assigned to that object is the sum of the credits allocated by each query to that object.

This finding has a nice property from the systems perspective: it suffices to run a local credit-maximizing scheduler on each participating device in order to achieve the overall objective. In general, local schedulers have the attractive property that they can locally adapt to bandwidth variations without coordinating with MSCloudQ, and need only minimal coordination with MSCloudQ in order to deal with new query arrivals. In MediaScope, the Object Uploader component of MSMobile implements the scheduling algorithm.

An Optimal Scheduler. We first describe a scheduling algorithm that is *optimal* under the assumption of fixed file sizes and fixed wireless bandwidth per participating device. Under these assumptions, for each object o, it is possible to compute the exact upload time $t(o)$ which is the same for all objects. If each object's timeliness bound is $d(o)$ (different objects can have different bounds), our goal is to find an uploading sequence such that $\sum_o g(o) \cdot c(o)$ is maximized.

First, we may assume that an optimal schedule orders the objects by earliest timeliness bound first. Assume an optimal schedule does not order objects by earliest timeliness bound first. Then there exist two objects i and j for which $d(o_i) >$

$d(o_j)$ but i is scheduled before j. By switching the order of objects i and j we can obtain another optimal schedule.

However, merely scheduling by earliest timeliness bound is not likely to maximize credit. To do this, the algorithm preprocesses the schedule to obtain a set of scheduled objects in the following way. It orders the objects by earliest timeliness bound first. Then, it adds objects to the schedule one right after another as long as each object's finish time does not exceed the timeliness bound. If an object's end time exceeds its timeliness bound, the algorithm removes the object receiving the smallest credit of those objects scheduled thus far (including current object) and shifts objects to the right of this object to the left by $t(o)$ to cover the gap. Intuitively, this step maximizes the total credit uploaded: lower credit objects, regardless of the query they belong to, are replaced. The algorithm then selects the next object in order of timeliness.

The following example illustrates this algorithm. Suppose there are 3 queries, each with one result object. Let their respective timeliness bounds be 2, 3, and 5 and the credits they receive be 7, 8, and 6 repectively. Finally, suppose $t(o)$ is 2 time units. The algorithm would proceed in the following way. It would schedule the first object initially. Since the second object would not be timely if scheduled after the first object and the second object receives more credits than the first, the first is removed and the second is scheduled from time 0-2. The third object is then scheduled from time 2-4 giving a maximal 14 total credits to the system.

Algorithm 2 : OPTIMAL UPLOADING SCHEDULE

Arrange the pending objects list $\mathbb{O}$ by earliest timeliness bound first, scheduling $\mathbb{S} \leftarrow []$
$l \leftarrow 0$
for $o \leftarrow \mathbb{O}.first$ **do**
  $\mathbb{S} \leftarrow o$
  $\mathbb{O}.remove(o)$
  if $l + t(o) \leq d(o)$ **then**
    $l \leftarrow l + t(o)$
  else
    Remove the smallest credited object in $\mathbb{S}$
    Shift all objects to the right of this object to left by $t(o)$

OUTPUT: scheduling $\mathbb{S}$, uploading object $\mathbb{S}[0]$

This algorithm is a special case of an optimal pseudo-polynomial algorithm discussed below, so we omit a proof of its optimality.

Optimality under different object sizes. If object uploading times are different, the scheduling problem is NP-hard; the simple case of different object sizes with all objects having the same timeliness bound is equivalent to the NP-Hard Knapsack problem [17]. We can however give the following pseudo-polynomial time dynamic programming algorithm for this problem. Let $S[i, q]$ be the maximum credited schedule using only the first i objects, i.e., objects $o_1, \ldots, o_i$, taking up q time units. Let $s[i, q]$ be the corresponding credit for such a schedule. Then $s[i, q]$ is defined in the following way:

$$s[i,q] = \begin{cases} \max\{s[i-1, q-t(o_i)] + c(o_i), s[i-1,q]\} & \text{if } q \leq d(o_i) \\ s[i-1,q] & \text{if } q > d(o_i), \end{cases} \quad (6)$$

where the following initial conditions hold: $s[0, q] = s[i, q < t(o_1)] = 0$. If $s[i-1, q-t(o_i)] + c(o_i) > s[i-1, q]$ and $q \leq d(o_i)$, then $S[i, q] \leftarrow S[i-1, q-t(o_i)] \cup \{o_i\}$, else $S[i, q] \leftarrow S[i-1, q]$. The desired output is $S(n, d(o_n))$ for an input of n objects.

The running time of this algorithm is $O(nd(o_n))$. The optimality of Algorithm 2 follows from the optimality of this dynamic programming algorithm for the general case [8].

Practical Considerations. In a practical system, the Object Uploader estimates $t(o)$ continuously, and re-computes the schedule after each upload is completed, in order to determine the next object to upload. There are two reasons for this. First, $t(o)$ can change because available wireless bandwidth can vary. Second, new queries may arrive at MSCloud; when a query arrives, MSCloud evaluates the query, assigns credits to the query results, and notifies the relevant devices (those which contain one or more result objects). Thus, at a given device, the set of objects to be uploaded can vary dynamically, so the Object Uploader needs to re-evaluate the schedule after every upload. Finally, for large objects, bandwidth variability might cause their timeliness bounds to be violated (e.g., because the available bandwidth became lower than the value that was used to compute the schedule); in this case, the Uploader can abort in-progress transmission, and we have left this optimization to future work.

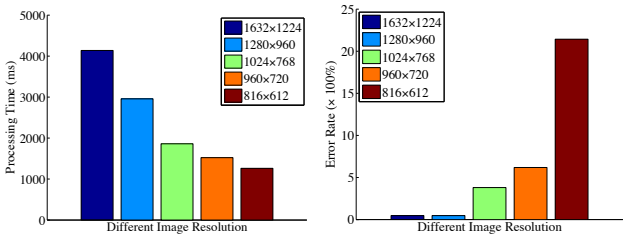
3.2.3 Feature extraction on the phone

In MediaScope, feature extraction is performed on the mobile device by the Feature Extractor component. This component extracts features for photos, as well as images extracted from videos. Even for high-end smartphone platforms, these are nontrivial computation tasks and some computation vs. accuracy trade-offs are required in order to achieve good performance. We now discuss these trade-offs.

Image Feature Extraction. The Samsung Galaxy S III (a high-end smartphone at the time of writing) can generate images with native resolution of 3264x2448. At this resolution, our CEDD feature extraction algorithm fails because of lack of memory on the device. One way to overcome this limitation is to resize the image to a smaller size and compute features on the smaller image.

As Figure 4(a) shows, the time to compute features (averaged over 300 images taken on the Galaxy SIII) can reduce significantly for different sizes, ranging from 4s for a resolution about 1/2 the native resolution to about 1s for 1/4 the native resolution. The cost of the resizing operation itself is about 250ms, roughly independent of the resized image size (omitted for brevity).

However, computing features on a smaller image trades off accuracy for reduced computation time. To explore this



(a) Average CEDD Execution Time Per Image for different SIZE

(b) Average Error Rate of KMeans Clustering for different Size

Figure 4—Image Resizing Overhead and Tradeoffs

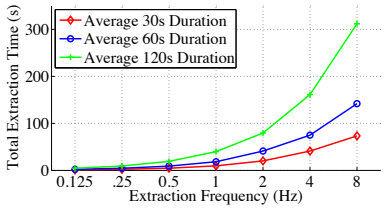


Figure 5—Average Video Frame Extraction Time For Different Duration and Frequency

trade-off, we evaluated two queries to see how accuracy varies with resizing. Figure 4(b) shows the results for K-means clustering, whose error rate is obtained by dividing the total number mis-classified images by the total number of images. This error rate is less than 5% for a 1280x768 resolution, but jumps to 20% for the 816x612 resolution. The error rate for K-nearest neighbor queries is defined as the ratio of incorrect images (relative to the the full size) selected by feature vectors computed on a resized image and k, averaged over different values of k. In this case, the knee of the error curve occurs somewhere in between the resolution of 1280x960 and 1024x768 (figure omitted for space).

Given these results, we use a resizing resolution of 1024x768 in our implementation as the best trade-off between computation time and accuracy.

Video frame extraction. The second major component of MSMobile's Feature Extractor is video frame extraction. Ideally, for videos, we would like to be able to extract every frame of the video and compute features for it. This turns out also to be computationally infeasible even on a high-end device, and one must perform a computation accuracy trade-off here as well, by subsampling the video to extract frames at a lower rate than full-motion video.

Figure 5 shows the total cost of frame extraction for videos

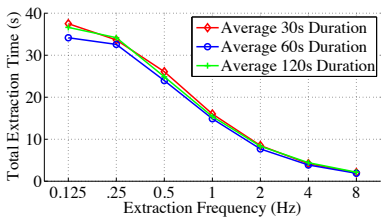


Figure 6—Average Inter-frame Feature-Space Distance

of different durations. Clearly, for long videos, even are relatively modest sampling rate of 4 fps can incur a total processing time of 150 seconds! On the other hand, extracting a single frame takes on average 240 ms, regardless of frame rate or duration.

On the flip side, subsampling a video can introduce errors; successive frames, if they are far apart from each other, may miss important intervening content. Figure 6 shows the average distance in feature space between successive frames for videos of different durations and sampling frequencies. For context, our clustering algorithms have generally found that cluster diameters are at least about 20 units. At 0.5fps, the interframe distance is more than this number, but at 1 fps, it is less. More generally, 1 fps seems to be a good choice in the trade-off between computation time and accuracy, so our current prototype uses this value.

An alternative approach to feature extraction for videos would have been to *segment* a video on the mobile device and then select frames from within the segment. A segment roughly corresponds to a scene, so one might expect that frames within a segment might have similar feature vectors. We have left an exploration of this to future work.

3.2.4 Leveraging a Crowd-Sensing Framework

MediaScope leverages an existing, publicly available, crowd sensing programming framework called Medusa [25]. Medusa provides high-level abstractions for specifying the steps required to complete a crowd-sensing task: in our case, uploading the feature vectors can be modeled as a crowd-sensing task and so can the upload of selected media objects. Medusa employs a distributed runtime system that coordinates the execution of these tasks between mobile devices and a cloud service. In MediaScope, MSCloud uses Medusa to distribute tasks and collect the results; MSMobile consists of extensions to Medusa's runtime to implement the Feature Extractor and the Object Uploader.

However, in order to support MediaScope, we needed to extend the Medusa model, which was focused on tasks generated by human users. We also needed to make several performance modifications in Medusa. In the former category, we modified Medusa's programming language to selectively disable Medusa's recruitment feature and data privacy opt-in: these features require human interaction, and MediaScope assumes that participants have been recruited and have signed a privacy policy out-of-band. Furthermore, we also added a data delivery notification system that would allow Medusa's cloud runtime to deliver notification of data upload to external servers, such as MSCloudDB. In the second category, we modified Medusa's mobile device notification system, which originally used SMSs, to use Google's C2DM notification service, which greatly reduced the latency of task initiation on mobile devices. We also optimized several polling loops in Medusa to be interrupt-driven, so that we could hand-off data quickly to components within Medusa's runtime as well as to external servers.

4. EVALUATION

In this section, we evaluate the performance of MediaScope. Although MediaScope's credit assignment algorithm is optimal in a pseudo-polynomial sense, we are interested in its practical performance under bandwidth variability. Furthermore, in practice, since query arrival cannot be predicted ahead of time, the practical performance of MediaScope may deviate from the optimal. Finally, it is instructive to examine alternative scheduling mechanisms to quantify the performance benefits of MediaScope's algorithms. We are also interested in the overhead imposed by MediaScope; since timeliness is an essential attribute of many queries, system inefficiencies can impact query completeness.

All our experiments are conducted on a prototype of MediaScope. MSCloud is written mainly in Python; PHP and Python are used for MSCloudQ web interface. The implementation of MSCloud is about 4300 lines of PHP and Python code, and MSMobile requires about 1150 lines of C and Java code (measured using SLOCCount [30]).

Our experiments use commodity hardware, both for MSCloud and the mobile device. We use up to 8 Android phones, which are either the Galaxy Nexus or the Galaxy SIII. MSCloud runs on a Dell XPS 7100 (six-core AMD Phenom II X6 1055T 2.8 GHz processor and 6MB built-in cache).

Before describing our results, we give the reader some visual intuition for the usefulness of MediaScope. Figures 7, 8, and 9 show the results of three different queries: a K nearest neighbor query, a Cluster Representatives query and a Spanner on a set of six groups of photos: a university campus, a garden, a view of the sky framed by trees, an athletics track, a supermarket, and a laboratory. Notice that the cluster representatives query identifies representatives from each of groups, while the Spanner extracts qualitatively different pictures, while the K nearest neighbor query extracts matching images as we might expect.

4.1 Query Completeness

In this section, we evaluate query completeness in the presence of concurrent queries.

Metrics and Methodology. Our metric for query completeness is the total credit associated with all the query results successfully uploaded before their timeliness bounds. We evaluate several *query mixes* (described below), with different concurrent queries of query types that arrive at different times and have different timeliness bounds. These queries are all posed on 320 images captured on 8 mobile devices.

Our experiments are conducted as follows. For each query mix, we first compute the results of each query and the credit assigned to each result object. This computation yields a *trace*, on each mobile device, of objects, their associated credits and the arrival time. We use this trace to replay the credit-based scheduling algorithm during repeated runs and report the average of 10 runs.

This trace-based methodology is also useful in comparing MediaScope's credit-based scheduling algorithm (henceforth, *MSC*) with several alternatives. For each alternative, we replay the trace for that particular scheduling algorithm. We consider the following alternatives: an *Omniscient* algorithm that knows about future query arrivals; a *Max Credit First (MCF)* that always selects the object with a maximum credit to upload; a *Round Robin (RR)* that allocates bandwidth fairly to each concurrent query so that, in each round, the object with the highest credit from each query is uploaded; and an *Earliest Deadline First (EDF)* scheduler that always schedules that object with the earliest timeliness bound first, breaking ties by credit. The Omniscient algorithm demonstrates the benefits of lookahead, while each of the other algorithms has at most one of MSC's features (timeliness-, credit-, and bandwidth-awareness).

In our experiments, each mobile device contains a number of images taken with its camera. These images are naturally of different sizes because they have different levels of compressibility. Furthermore, we make no attempt to control network variability; upload bandwidths in our experiments vary and MSC estimates upload bandwidth by measuring the average speed of the last upload (MSC's algorithm needs uses this estimate for $t(o)$).

Results. Our first experiment compares the performance of these alternatives for three different query mixes with different types of queries. The first mix contains 4 queries, namely, 1 Top-K, 1 Spanner, 1 Cluster Representative and 1 Common Interest. All the queries arrive at the same time but with different timeliness bounds; thus, in this experiment there are no future arrivals and we do not evaluate the Omniscient algorithm. The second mix adds one more Cluster Representative query to the first one, and the third is generated by adding one more Common Interest query. In each query mix, each query is assigned the same total credit.

Figure 10 shows the performance of various schemes. MSC achieves at least 75% completeness across all three query mixes, and its performance improves by 5% as the number of queries increases from 4 to 6. Although a 75% completeness rate seems pessimistic, we remind that reader than MSC is optimal, *so no other scheduling scheme could have done better than this*; in other words, for this query mix, this is the best result that could have been achieved.

Furthermore, MSC outperforms other schemes significantly. The superior performance of MSC comes from three factors: it is timeliness-aware, credit-aware, and also adapts to available bandwidth. By contrast, approaches that lack one or more of the features have much lower completeness rates. Thus, EDF does not take into account an object's credit, and thus might waste bandwidth on objects with an early deadline but small credit; on average, EDF achieves 55% completeness.RR is unaware of timeliness constraints, but uploads the result objects for each query in a round-robin fashion. It is comparable in performance to EDF, achieving 52% completeness on average. RR's poor performance

Figure 7—K Nearest Neighbor Result

Figure 8—Cluster Representative

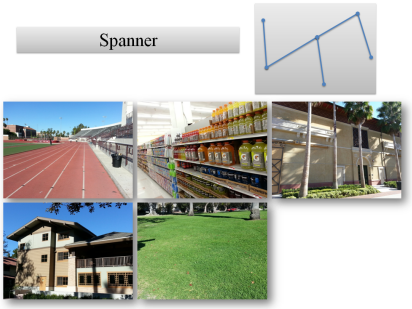


Figure 9—Spanner

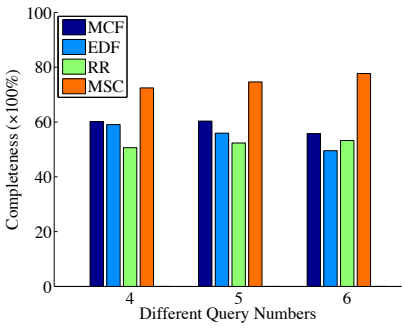


Figure 10—Different Query Mixes by Size

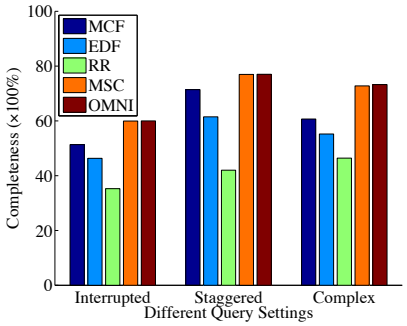


Figure 11—Different Query Mixes by Timeliness Bound

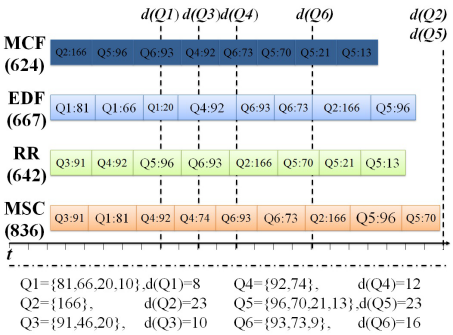


Figure 12—Sample Schedule Timeline

arises from two factors: first, because it ignores timeliness constraints, it uses transmission opportunities by sometimes transmitting objects which could have been deferred without violating data timeliness bounds; second, RR gives equal transmission opportunities to queries, even though, on a given mobile device, one query may contain objects with far more credit than another query. MCF improves upon RR in the second aspect, in that it always transmits the object with the highest credit first; in so doing, it achieves an average completion rate of 59% and is significantly better than EDF and RR. However, MCF is still noticeably worse than MSC, primarily because MCF ignores timeliness constraints and sometimes transmits objects that could have been deferred without violating timeliness bounds.

In order to get more insight into the relative performance of these schemes, we consider variants of the 6-query mix which have different combinations of arrival rates and deadlines. Figure 11 plots the results of these experiments.

In the first query mix, three of the six queries arrive first with the timeliness bound of 20 seconds. The remaining three queries arrive within three seconds, but have a relatively tight timeliness bound off 6 seconds. In this sense, they *interrupt* the first set of queries. This query mix is designed to demonstrate the benefits of timeliness-awareness. In this somewhat adversarial setting, MSC still outperforms other schemes but has a much lower completeness rate of about 60%. RR performs poorly, but EDF performs comparably to MCF; this is not surprising because EDF is timeliness-aware. Even so, EDF does not perform as well as MSC because it ignores credit values and uploads objects with lower credits unnecessarily.

In the second query mix, 6 queries with the same timeliness requirement arrive in a staggered fashion, with each query arriving three seconds after the previous query. This illustrates a setting where queries arrive frequently but the arrivals are not synchronized. In this setting, MSC achieves a completeness rate of nearly 80%, and, not surprisingly, MCF comes quite close with a completeness rate of 71%. Since all queries have identical timeliness bounds, it is not surprising that a credit-aware scheme like MCF performs well.

The third query mix represents a complex pattern where queries arrive at different times and have different deadlines. For this mix, the performance advantages of MSC are clear, since this mix requires a scheduling scheme to be both credit and timeliness-aware.

Finally, for all these query mixes (Figure 11, MSC is comparable to the Omniscient scheme, which knows the arrival times of different queries. Intuitively, because MSC continuously adapts its transmission schedules when new queries arrive, it can make a different decision from Omniscient only at the times when queries arrive. To be more precise, say a new query arrives at time t: Omniscient might have scheduled an upload of an object for the new query starting at time t, but MSC has to wait until the object being uploaded at t finishes, before it updates its schedule. This difference can be fixed by adding *preemption* to the scheduler, aborting the current transmission if it does not have the highest priority; we have left this to future work.

Components	Average Latency(ms)
MSCloud to Medusa	131
C2DM (send-to-receive)	150
Task execution	67
Upload scheduling	46
Medusa to MSCloud image transfer	67

Table 1—Potential Drawback Factors For uploading Speed

System Components	Average Latency(ms)
Query Parsing	24
Feature Vector Download	138
Medusa Server Interpretation	68
Spanner	89
K Clusters	52
K Nearest Neighbor	11
Query Result Response	54

Table 2—System Components Overhead

To get some more insight into the differences between the scheduling algorithms, Figure 12 plots the timeline of decisions made by these algorithms for the 6-query mix when all queries arrive at the same time. The figure clearly shows that MSC is better able to use the available time to carefully schedule uploads so that completeness is maximized; MCF, having uploaded objects with high credits is unable to utilize the available time because the timeliness bound for the remaining objects has passed. EDF performs comparably to MCF, but, because it is credit-unaware, misses out on some transmission opportunities relative to MSC (e.g., MSC uploads Q3:91 first, but EDF does not).

4.2 System Overhead

Latency. Because MediaScope attempts to satisfy timeliness constraints, the efficiency of its implementation can impact query completeness; the less overhead incurred within the system, the greater the query completeness can be. To understand the efficiency of our system, we profiled the delays within the various components of MediaScope (Table 1). In an earlier section, we have discussed the cost of feature and frame extraction: these operations are not performed in the object retrieval path, so do not affect query timeliness.

As this table shows, the latency incurred for most components is modest; C2DM notifications take about a 1/6th of a second, and the communication between MSCloud and Medusa takes 1/5th. Other components are under 70 ms.

Finally, latency within the MSCloudQ query engine is also moderate (Table 2). Even in our relatively un-optimized implementation, most components of query processing take less than 100ms, with the only exception being the download of feature vectors from MSCloudDB; we plan to optimize this component by caching feature vectors in memory.

These overhead numbers suggest that our current prototype may be able to sustain timeliness bounds of 10s or lower. Indeed, some of our experiments in the previous section have used 6s timeliness bounds.

Energy. The other component of overhead is energy expenditure. Frame extraction and feature extraction can take up to a second, or more, of CPU time. The energy cost, on a Motorola Droid (measured using a power meter), of frame extraction is 57 μAh, and of feature extraction (including resizing) is 331 μAh. We believe these energy costs are still reasonable: for feature extraction to consume even 10% of the Droid's battery capacity, a user would have to take nearly 400 photos!

5. RELATED WORK

Perhaps the closest related piece of work to MediaScope is CrowdSearch[31], which attempts to search for the closest match image generated on a mobile device from among a set of images stored on a photo sharing service. Its focus, however, is complementary to MediaScope, and is on bridging the semantic gap inherent in feature-based image searches; most feature extraction methods do not understand the semantics of images, and CrowdSearch focuses on using human intelligence in near real-time to complete search tasks. MediaScope can use this capability to filter search results to bridge the semantic gap, but its focus is on supporting a richer query interface and enabling tighter timeliness constraints than might be possible with humans in the loop.

Also closely related is PhotoNet [27], which proposes an opportunistic image sharing and transmission capability in a delay tolerant network. PhotoNet uses similar image features to perform photo comparisons, but is otherwise very different from MediaScope in that the latter explicitly supports a query interface with timeliness constraints on queries.

MediaScope is informed and inspired by several pieces of work on techniques for content-based image retrieval, and image search on mobile devices.

In the former category are systems like Faceted Image Search [33], the Virage Image Search Engine[7] and ImgSeek [19], that support searches on a centralized database of images. MediaScope builds upon these search techniques, but unlike them, supports timely geometric queries over a distributed database of images and videos on mobile devices. Other work in content-based image retrieval has proposed clustering [9, 13], but has not explored the mobile device setting.

A second category of work has explored support for image searches on a mobile device. For example, [21] discusses energy efficient feature extraction on a mobile device but supports on the local searches on the device, as does [32]. Other pieces of work have explored a client/server architecture for image search, but where the content is stored on the server [20, 16, 6]. By contrast, MediaScope supports searches on a cloud server, but where the content is stored on the mobile devices and is retrieved on demand.

Finally, tangentially related to MediaScope is work on automated or semi-automated annotation of images with context obtained from sensors [15, 28, 24]. MediaScope can use such annotations to support a broader range of queries, but we have left this to future work.

6. CONCLUSIONS

In this paper, we have discussed the MediaScope, a system that bridges the availability gap for visual media by supporting timely on-demand retrieval of images and video. MediaScope uses a credit-based timeliness-aware scheduling algorithm that optimizes query completeness, and its overheads are moderate. Much work remains including optimizing the internals of the system to improve completeness, and supporting more geometric queries on visual media.

7. REFERENCES

[1] Cops using youtube to catch criminals. *http://www.afterdawn.com/news/article.cfm/2007/03/04/cops_using_youtube_to_catch_criminals*.

[2] Facebook. http://www.facebook.com.

[3] Facebook. http://www.instagram.com.

[4] Flickr. http://www.flickr.com.

[5] Teen titan. *http://www.newyorker.com/reporting/2012/09/03/120903fa_fact_widdicombe*.

[6] I. Ahmad, S. Abdullah, S. Kiranyaz, and M. Gabbouj. Content-based image retrieval on mobile devices. In *Proc. of SPIE*, volume 5684, pages 255–264, 2005.

[7] J. Bach, C. Fuller, A. Gupta, A. Hampapur, B. Horowitz, R. Humphrey, R. Jain, and C. Shu. The virage image search engine: An open framework for image management. In *SPIE Storage and Retrieval for Image and Video Databases IV*, pages 76–87, 1996.

[8] J. Blazewicz, K. H. Ecker, E. Pesch, G. Schmidt, and J. Weglarz. *Handbook on Scheduling: From Theory to Applications*. Springer, 2007.

[9] D. Cai, X. He, Z. Li, W. Ma, and J. Wen. Hierarchical clustering of www image search results using visual, textual and link information. In *Proc. of the 12th annual ACM international conference on Multimedia*, pages 952–959. ACM, 2004.

[10] S. Chatzichristofis and Y. Boutalis. Cedd: color and edge directivity descriptor: a compact descriptor for image indexing and retrieval. *Computer Vision Systems*, pages 312–322, 2008.

[11] S. Chatzichristofis and Y. Boutalis. Fcth: Fuzzy color and texture histogram-a low level feature for accurate image retrieval. In *Ninth International Workshop on Image Analysis for Multimedia Interactive Services. WIAMIS'08.*, pages 191–196. IEEE, 2008.

[12] S. Chatzichristofis, Y. Boutalis, and M. Lux. Selection of the proper compact composite descriptor for improving content based image retrieval. In *Proc. of the 6th IASTED International Conference*, volume 134643, page 064, 2009.

[13] Y. Chen, J. Wang, and R. Krovetz. Content-based image retrieval by clustering. In *Proc. of the 5th ACM SIGMM international workshop on Multimedia information retrieval*, pages 193–200. ACM, 2003.

[14] Z. Chi, H. Yan, and T. Phážąm. *Fuzzy algorithms: with applications to image processing and pattern recognition*, volume 10. World Scientific Publishing Company Incorporated, 1996.

[15] M. Davis, N. V. House, J. Towle, S. King, S. Ahern, C. Burgener, D. Perkel, M. Finn, V. Viswanathan, and M. Rothenberg. Mmm2: mobile media metadata for media sharing. In *Proc. of CHI'05 extended abstracts on Human factors in computing systems*, pages 1335–1338. ACM, 2005.

[16] M. Gabbouj, I. Ahmad, M. Amin, and S. Kiranyaz. Content-based image retrieval for connected mobile devices. In *Proc. of Second International Symposium on Communications, Control and Signal Processing (ISCCSP)*. Citeseer, 2006.

[17] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman and Company, 1979.

[18] J. Huang, S. Kumar, M. Mitra, W. Zhu, and R. Zabih. Image indexing using color correlograms. In *Proc. of IEEE Computer Society Conference on Computer Vision and Pattern Recognition(CVPR'97)*, pages 762–768. IEEE, 1997.

[19] C. Jacobs, A. Finkelstein, and D. Salesin. Fast multiresolution image querying. In *Proc. of the 22nd annual conference on Computer graphics and interactive techniques*, pages 277–286. ACM, 1995.

[20] J.S.Hare and P. Lewis. Content-based image retrieval using a mobile device as a novel interface. In *Electronic Imaging 2005*, pages 64–75. International Society for Optics and Photonics, 2005.

[21] K. Kumar, Y. Nimmagadda, Y. Hong, and Y. Lu. Energy conservation by adaptive feature loading for mobile content-based image retrieval. In *ACM/IEEE International Symposium on Low Power Electronics and Design (ISLPED'08)*, pages 153–158. IEEE, 2008.

[22] A. Lipkus. A proof of the triangle inequality for the tanimoto distance. *Journal of Mathematical Chemistry*, 26(1):263–265, 1999.

[23] M. Lux and S. Chatzichristofis. Lire: lucene image retrieval: an extensible java cbir library. In *Proceeding of the 16th ACM international conference on Multimedia*, pages 1085–1088. ACM, 2008.

[24] C. Qin, X. Bao, R. R. Choudhury, and S. Nelakuditi. Tagsense: a smartphone-based approach to automatic image tagging. In *Proc. of the 9th international conference on Mobile systems, applications, and services(Mobisys'11)*, pages 1–14. ACM, 2011.

[25] M. Ra, B. Liu, T. L. Porta, and R. Govindan. Medusa: A programming framework for crowd-sensing applications. In *Proc. of the 10th international conference on Mobile systems, applications, and services(Mobisys'12)*, pages 337–350. ACM, 2012.

[26] T. Tanimoto. *An elementary mathematical theory of classification and prediction*. International Business Machines Corporation, 1958.

[27] M. Uddin, H. Wang, F. Saremi, G. Qi, T. Abdelzaher, and T. Huang. Photonet: a similarity-aware picture delivery service for situation awareness. In *IEEE 32nd Real-Time Systems Symposium (RTSS'11)*, pages 317–326. IEEE, 2011.

[28] W. Viana, J. B. Filho, J. Gensel, M. Villanova-Oliver, , and H. Martin. Photomap: from location and time to context-aware photo annotations. *Journal of Location Based Services*, 2(3):211–235, 2008.

[29] C. Wang, L. Zhang, and H.J.Zhang. Learning to reduce the semantic gap in web image retrieval and annotation. In *Proc. of the 31st annual international ACM SIGIR conference on Research and development in information retrieval*, pages 355–362, 2008.

[30] D. Wheeler. Sloccount, 2001.

[31] T. Yan, V. Kumar, and D. Ganesan. Crowdsearch: exploiting crowds for accurate real-time image search on mobile phones. In *Proc. of the 8th international conference on Mobile systems, applications, and services(Mobisys'10)*, pages 77–90. ACM, 2010.

[32] J. Yang, S. Park, H. Seong, H. Byun, and Y. Lim. A fast image retrieval system using index lookup table on mobile device. In *19th International Conference on Pattern Recognition(ICPR'08)*, pages 1–4. IEEE, 2008.

[33] K. Yee, K. Swearingen, K. Li, and M. Hearst. Faceted metadata for image search and browsing. In *Proc. of the SIGCHI conference on Human factors in computing systems*, pages 401–408. ACM, 2003.