Good afternoon. Today I’m gonna talk about an interesting system, called MediaScope. With MediaScope, it’s possible to search and retrieve media contents captured by phones, even the media contents that recently captured and had not been uploaded to anywhere.

Mobile devices, like smartphones, are nowadays equipped with high-end camera, which can capture images with high resolution and great quality. As a result, people like to capture visual media by mobile devices, and share with others through | social network like Facebook, Twitter, or dedicated image sharing service like flickr or instagram. | However, such upload is usually not immediate, and its future trend will be more and more delayed. for 2 reasons, first, the increasing bandwidth requirements introduced by high-resolution cameras is far faster than the cellular capacity; second, cellular data plan currently has a restricted usage limit. Such 2 reasons restrict immediate uploads, and lead to an availability gap for newly captured images.

Actually, the availability gap already exists. We did a study of Flickr uploads, by randomly selecting 40 users, each with 50 images. | We calculated the availability gap for each image and found that, 50% of the images are uploaded more than 10 days after they are captured. Such availability gap means that the images we currently view are not recent.

Assuming if we can bridge the availability gap, in other words, we can view all the images captured by any mobile devices. Such capability can be very useful in lots of scenarios. | e.g., an image captured by a passed-by customer can help resolve mall robbery; | in a sporting event, the sportwritter wants the best view of a dunk, sometimes captured by audience with mobile device.

Numerous other scenarios can benefit from bridging the availability gap. But how can we bridge the gap? | If we view each mobile device as image storage, then all the mobile devices form a worldwide distributed database of the most recent and diverse media contents. One way to bridge the availability gap is to support image retrieval on this image database.

We design a system, called MediaScope, to realize timely on demand media retrieval from mobile devices. | On-demand retrieval bridges the availability gap, allow us to retrieve newly captured images, for example, from today's basketball game; | timely retrieval is important for many applications, the mall security staff might want the retrieval to be finished within a short period of time, say, 30 seconds.

A high level picture of MediaScope approach is, MediaScope accepts search queries, | for example, images related to basketball, | it selects relevant images for this query and retrieves from corresponding mobile devices. | Of course there can be concurrent queries posed to MediaScope server, | here the other query interested in United States, and thus leads to concurrent retrieval from same mobile device. | MediaScope needs to timely retrieve images for multiple and maybe concurrent queries.

To retrieve images, MediaScope supports queries for different retrieval requirements. | For example, a Top-K query retrieves K most similar images to the target image. A sportwritter can use this query to find desired sporting event image. Another example, | the spanner query, returns a collection of most dissimilar images. Spanner query can help the mall security staff to understand the range of available images during the robbery. Later we will see MediaScope extensibly supports more queries.

To realize this, MediaScope faces several challenges. First, we must select relevant query results to retrieve. Image search can be very computationally expensive; MediaScope adapts and generalizes techniques from content-based image retrieval. Second, query results must be returned in a timely manner, which is hard, given variable and limited wireless bandwidth. MediaScope addresses this by optimizing uploaded information.

Inspired by image search techniques that support similarity search on image feature space, MediaScope selects relevant images based on not only metadata, like location and time information of images, | but also feature vectors reflects the visual information.

Generally speaking, a feature vector is a series of coefficients, depicts statistical summary of some visual information, for instance, color histogram. The basis of similarity search is to define a similarity metric on 2 feature vectors to show whether these 2 images are similar or not. A popular way to do this is to view an n coefficients feature vector, | as a point in n-dimensional space and then use the euclidean distance to define similarity. Based on this similarity metric, MediaScope defines more complicated geometric queries than traditional image search, to support different retrieval requirements.

(rest) MediaScope system is partitioned across a cloud component called MSCloud | and a mobile devices' component, called MSMobile. | It leverages a publicly available crowd sensing platform called Medusa to communicate with mobile devices. | MSCloud periodically initiates feature extraction task, then newly captured images' features will be uploaded to MSCloudDataBase. | Users pose queries to MSCloud web interface, MSCloud selects relevant query results based on feature vectors stored in MSCloudDataBase. | Then it cooridinates with MSMobile's Object uploader component to realize the retrieval of selected images. In the following, we will focus on three challenging aspects of MediaScope, | an accurate yet efficient feature extraction method; select relevant results for different geometric queries and the support for concurrent queries.

The feature extractor on MSMobile side, adopts widely accepted color and edge directivity descriptor as feature vector, which contains color and edge information in a 144 coefficients ranging from 0~7, makes the feature vector as small as 54 bytes for 1 image. However, image generated by current mobile devices are with high resolution over 6 megapixels, makes calculating feature vector a resource consuming task, sometimes even lead to out of memory error. | To tackle this, MSMobile resize the images before extracting features for shorter extracting time. | But, this method introduces some classification error. We empirically find the | sweet spot of resizing resolution, which reduces the extraction time to roughly 1 second per image with less than 4% error rate.

On this image feature space, MediaScope supports geometric queries. For illustration, in these graphs we assume a simplified 2 dimension feature space; each vertex represents an image feature vector. | We define similarity function, with greater value for more similar feature vector pair and list 3 geometric query examples that MediaScope supports. | top-K query returns k images with maximum similarity value to the target image, we have a query image here, and the query results might be these 2. | Spanner query tries to minimize the maximum similarity among selected images in order to get most dissimilar images. The 3rd query, | the cluster representative query, clustering the images first, and select the best representative for each cluster, which is the one with the maximum similarity value to the center of that cluster. Any other geometric query on feature space can be extensibly implemented in MediaScope.

The most challenging component of MediaScope is the support for concurrent queries. MSCloud may receive one or more queries while other queries are being processed. | Consider the simplest case where we have 2 mobile devices. | We got Query 1, asking for some images from these 2 phones, | at the same time we have Query 2. | Assume these 2 queries have the same timeliness bound, 5 time units, and uploading any image needs 1 unit. It's clear that each single query can be satisfied, but we cannot satisfy both of them. | MediaScope tackles this bandwidth scarcity issue by trading off query completeness for timeliness, namely, to maximize the amount of information retrieved for each query. To do so, MediaScope uses a credit assignment mechanism.

Each query is assigned by credits and then divides up these credits among the selected query results in a way that reflects the importance of each selected image to the query. | The intuition of this dividing up is the importance of an image to the query can be reflected by the feature space geometry. | For example, a Top-K query with 1000 credits selected 3 images. | Naturally, the more similar to the target image, the more credit that image should be assigned, | thus we can divide 1000 credits like this. | Spanner query wants the most dissimilar images, and thus an important image should be the one that is quite dissimilar to others.

Given assigned credits, the goal of MediaScope is clearly to maximize uploaded credits for each query. When there are concurrent queries, | MediaScope tries to maximize the sum of credits of each query. | For example, assume all the selected images have been uploaded in this case, the objective function MediaScope tries to maximize, is by definition the sum of uploaded credits from each query | this is from query 1, this is from query 2; but this objective function can also be represented as the sum of uploaded credits for each mobile device,| you can see, this is from phone 1, and phone 2.

Why is this property useful? It means that, instead of | a global scheduler located at MSCloud tries to maximize the sum of each query's credits, MediaScope can place the scheduler to each | mobile device and let it to optimize its own uploading independently. Let's see which solution is better. | the global scheduler is quite complicated, | which must take into account all queries information, | especially each phone's instant uploading status, like uploading speed; get such information and then remotely control the uploading is challenging, and sometimes infeasible for delay-sensitive tasks like timely retrieval. However, | a local scheduler at each phone is much simpler. | At any time, each phone has a list of pending selected images, associated with its filesize, which can be used to estimate its uploading time, the credit of this image and its timeliness bound. The local scheduler doesn't care about which query each image belongs to, it only cares about whether the upload is on time or not, an on-time upload gains associated credits. | With instant uploading status available locally, apparently, it's better to place the scheduler at each mobile device.

Scheduler at phone side has a list of pending images, with filesize, credit and deadline. We provide an optimal scheduling algorithm, for the case that images are with the same size and thus same uploading time. Such assumption is not impractical since for each phone device it captures images with similar sizes. The algorithm tries to arrange pending images in an earliest deadline first manner, and always throw away the image with smallest credit when the arrangement cannot continue. | In this case, we have 5 objects with 2 different timeliness bounds. three of them have deadline at time 3, the other two are with earlier deadline of time 2. | We arrange them by the order of deadline, | these 2 objects with earlier deadline are scheduled for first and second slots; then we consider the other 3 objects with later deadline, | first object with 500 credits can fit in; however, the second one with also large credit, 300, cannot fit in. | We throw away the object with smallest credit, in this case this 100 credit image, and continue the process. We prove this algorithm always return the optimal results. With different file sizes, the problem becomes NP-Hard, but we can still use dynamic programming to get the optimal uploading schedule for small cases.

We implement MediaScope prototype. In MSCloud, the query server is written in PHP and Python, MSCloud database is using mysql; at mobile device side, MSMobile is developed in Java. | Our evaluation experiment involves 8 android phones. The main metric we are using is the query completeness, namely, the average portion of uploaded credits for all the queries. Due to instable wireless bandwidth, we repeat the experiments of the same query input for multiple times.

We compared our MediaScope credit-based algorithm, MSC, to several alternatives. | Max credit first scheme always upload image with maximum credit; | earliest deadline first scheme uploads objects with the tightest deadline; | a round-robin scheme uploads selected images for each query one by one to achieve fairness among queries. A special one, omniscient algorithm, which knows the future incoming queries, serves as the upper bound of the optimal solution. | Our first experiment contains 3 test cases, | with 4, 5, 6 concurrent queries, all the queries are with the same starting time and timeliness bound. | We see MSC outperforms other schemes in query completeness, | with larger improvement ratio when there are more concurrent queries. The second experiment tested three types of query mix. | Interrupted query mix contains 3 queries with late timeliness bound, however, later another 3 queries with earlier deadlines interrupt with existing queries; interrupted query mix is designed to test the deadline sensitivity of all the schemes. | A staggered setting, where 6 queries with the same timeliness bound arrives in a staggered manner, with each query arriving after the previous query, to simulate frequent query arrivals. | The third query mix represents a complex pattern where queries arrive at different times and have different deadlines. | For all three settings, MSC outperforms other schemes, since its both credit aware and deadline aware, | and also is comparable to the omniscient scheme.

We measured MediaScope's overhead in term of latency and energy. | We found that, except for images uploading, all other parts latencies are low. Since feature extraction takes 1 second CPU time, | we measured its energy consumption. | this number means that, 10% of the battery capacity can extract feature vectors of more than 400 images. We believe its consumption is reasonable

As the summary, we propose MediaScope, a timely on-demand media retrieval system from mobile devices. MediaScope accurately and efficiently extracts feature vectors of images, supports general geometric queries on it, timely returns informative retrieval results for queries; and finally, bridges the availability gap. Thank you.