Cloudlet-based Just-in-Time Indexing of IoT Video

Mahadev Satyanarayanan*, Phillip B. Gibbons*, Lily Mummert†, Padmanabhan Pillai‡, Pieter Simoens§, Rahul Sukthankar†
*Carnegie Mellon University
†Google
‡Intel Labs
§Ghent University, imec

***Abstract*—**As video cameras proliferate, the ability to scalably capture and search their data becomes important. Scalability is improved by performing video analytics on *cloudlets* at the edge of the Internet, and only shipping extracted index information and meta-data to the cloud. In this setting, we describe *interactive data exploration (IDE),* which refers to human-in-the-loop content-based retrospective search using predicates that may not have been part of any prior indexing. We also describe a new technique called *just-in-time indexing (JITI)* that improves response times in IDE.

***Keywords*-**image search, video search, interactive search, discard-based search, context-sensitive search, unindexed data search, edge computing, visual cloud, scalability, speculative processing

I. Ubiquitous Video Cameras

Always-on video cameras are proliferating in the Internet of Things (IoT). A 2013 survey in the U.K. estimated one surveillance camera in a public space for every 11 people [1]. Today, virtually every automobile in Russia has a video camera to record incidents for insurance purposes [2]. Extrapolating from these trends, a 2013 NSF report [3] predicts that "It will soon be possible to find a camera on every human body, in every room, on every street, and in every vehicle."

The video captured by these cameras is typically stored on local storage, close to the point of capture. It is examined only in response to some traumatic event such as a vehicular accident, a burglary, an accusation of police brutality, or a terrorist attack. *Without ever being examined, most data is overwritten* to reclaim space after a modest retention period. This represents an enormous loss of knowledge. Embedded in this data is information relevant to important questions that are hard to answer today. Can we extract this valuable information before discarding the raw data? For example, a lost child or pet may unexpectedly appear in video far from home. Timely recognition could lead to their rescue. As another example, video footage from road intersections could reveal those that have many near misses. Traffic lights or stop signs could then be installed in time to prevent serious accidents. As a third example, timely analysis of video from a sidewalk may reveal a number of people slipping on an icy patch that was missed by the salt crew. Prompt attention to the icy patch could avert a serious injury. As a final example, in marketing and sales, real-time video analytics could reveal that shoppers are ignoring a new window display. The richness of high-resolution video content and the open-endedness of deep video analytics make vision-based sensing especially attractive.

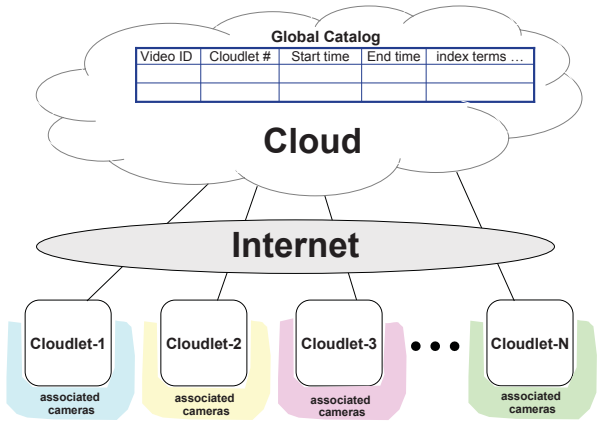

Fig. 1. Two-level Cloud-Cloudlet Architecture

II. Why Edge-based Video Analytics?

Video analytics is typically performed in the cloud today. Using Netflix's estimate of 3 GB per hour of HD video, one video stream demands nearly 6.8 Mbps. A 100 Gbps metropolitan area network (MAN) can only support about 15,000 such video streams. Even upgrading to a 1 Tbps MAN will only support 150,000 video cameras. Supporting a million cameras (one per home in a large city) will require nearly 7 Tbps. Shipping all video to the cloud is clearly not scalable.

Our solution is to process video close to the cameras, as shown in Figure 1. Below today's unmodified cloud is a second architectural level consisting of dispersed elements called *cloudlets* [4]. These have excellent network connectivity to associated cameras, sufficient compute power to perform video analytics, and ample storage to preserve video at full fidelity for a significant retention period before being overwritten. Extended retention permits retrospective search of captured video, as discussed in Section IV. Using the above figure of 3 GB for an hour of HD video, a single 4 TB disk that costs about $100 today could hold over 50 days of video from one camera. Only the results of video analytics (e.g., index terms and metadata such as cloudlet id and timestamp) are shipped to a global catalog in the cloud. Based on popularity and importance, small segments of full-fidelity video could also be shipped to the cloud for long-term archiving.

III. Background: Video Denaturing and Indexing

Each cloudlet in Figure 1 runs the *GigaSight* software for video processing. Since GigaSight has been described in a

978-1-5090-5873-0/17/$31.00

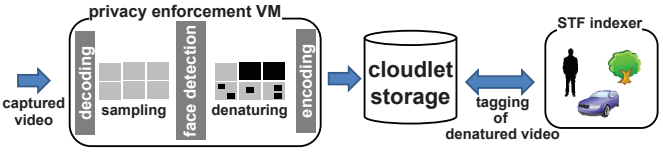

Fig. 2. GigaSight Video Processing Workflow on a Cloudlet

Fig. 3. Examples of STF Object Detection (Adapted from Shotton et al [5])

previous paper [6], we only provide a brief summary here as background to our new work in Sections IV–VII.

Privacy is a key concern of GigaSight. As shown in Figure 2, each cloudlet performs *denaturing,* which refers to automated, content-specific lowering of fidelity of video in order to preserve privacy. Isolation between video streams is ensured by performing the denaturing of each stream within its own virtual machine (VM). By default, GigaSight blurs all faces detected in video frames. However, under appropriate authorization controls, the original undenatured frames can be retrieved from its VM in order to support use cases such as looking for a lost child that require faces to be exposed. GigaSight performs content-based indexing of denatured video frames using Shotton et al's *Semantic Texton Forest (STF)* algorithm [5], with classifiers trained on the MSRC21 data set. This enables tagging of video frames with 21 classes of common objects such as aeroplanes, bicycles, birds, boats, etc. Figure 3 shows some example images along with the segmentation performed by STF. Extracted tags are propagated to the global catalog in the cloud (Figure 1) to support system-wide searches. GigaSight could easily be extended to use deep neural networks (DNNs) or other techniques for indexing.

To reduce cloudlet workload, GigaSight only processes periodic samples of frames from each video stream. The sampled frames effectively serve as "thumbnails" that are representative of content for the next N frames. Those next N frames are not denatured or indexed, but stored in encrypted form on the cloudlet. Those frames are only processed on demand, if their thumbnail triggers user interest (typically during during a search). A typical value of N is 300, thus giving one denatured and indexed frame every 10 seconds.

IV. Interactive Data Exploration

Cloudlet-based video capture, denaturing, and indexing as discussed in Sections II and III can only partially deliver the full value of video analytics as outlined in Section I. To complete the picture, we need a human-in-the-loop interactive image search capability that embodies the necessary flexibility and versatility to customize searches for the very specific needs of a user. To understand why, consider the hypothetical

> The owner of a dog has not seen her pet for 24 hours. A search on foot of the local neighborhood has yielded no results. Worried and anxious, the owner thinks her pet may have wandered off to some distant part of the city. She obtains permission from local authorities to search for her dog in denatured video on city cloudlets. Assuming a speed of two miles per hour (two-thirds that of humans), the dog could have wandered anywhere within an area of 12 square miles in 24 hours. That is half the size of Manhattan, and contains over 100,000 surveillance cameras outside residences and businesses (London is estimated to have 500,000 surveillance cameras today). In a 24-hour period, even if frames are denatured and indexed only once every 10 seconds on each video stream, there will be over 800 million frames to search. Although "dog" is one of the index terms supported by video indexing in cloudlets, the hit rate is too high: nearly one in every thousand indexed frames (0.1%) in this dog-friendly city has a dog somewhere in it. The index of the global catalog shrinks the search space from 800 million frames to 800,000 frames but that is still a daunting figure. The owner needs some tools to help search for *her* dog, not just any dog. Every hour of delay reduces the chances of rescuing her pet.

Fig. 4. Use Case: Searching for a Lost Dog

use case in Figure 4 of searching for a lost dog. This is exactly the kind of public service use case envisioned in Section I.

What kind of image search tools can we provide to help in scenarios such as Figure 4? The simplest answer would be to create a high-accuracy object detector for the lost dog using any of the well-known techniques today such as DNNs or SVMs, and then use it to index the subset of relevant frames on cloudlets. Unfortunately creation of an object detector requires a significant amount of training data, preferably hundreds or thousands of images. The pet owner may not have such a large number of images of her pet. She may have at most a few images, and in some cases she may not have any images at all. Yet, in her mind's eye, she has a very clear image of what her dog looks like.

If the dog is a pure-bred, perhaps there are pre-trained classifiers available for German Shepherds, Collies, Shetland Sheepdogs, etc. Using such a classifier to further narrow the search space would be a natural first step. The ability to introduce such a classifier easily in the course of a search, and to only index on demand a small relevant part of the whole dataset would be extremely valuable. If the dog is not a pure-bred or is a rare breed, no pre-trained classifier may be available. In that case, the owner may have to resort to more generic features such color or fur texture to narrow the search space. Ultimately, through some combination of search predicates that are obtained through trial and error on the actual data, the search space has to be narrowed down to human scale: i.e., a few hundred images that a user can manually scan carefully in a reasonable amount of time. If the owner is fortunate, her pet will be in one of the recently-captured video frames and the location of the relevant video camera will help to target her physical search.

In some cases, the search may be for a scene that is hypothesized to have occurred, with many particulars of the scene being fuzzy. For example, an insurance adjuster may want to verify a verbal accident report about a perpetrator

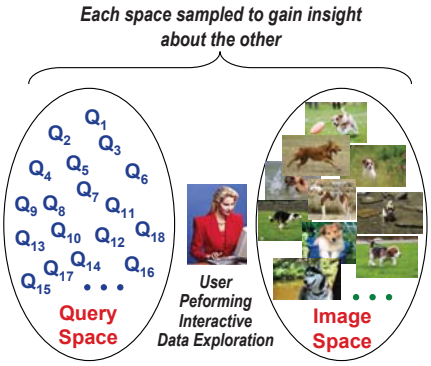

Fig. 5. IDE: Interleaved Search in Two Spaces

whose attributes are only vaguely known. It is only during the process of data exploration, after seeing many false positives and a few false negatives, that the search predicates themselves get refined. Unlike the previous example, where the target of the search (precise attributes of missing dog) was clear, even the target of the search may be fuzzy initially. The absence of training data for creation of detectors will be even more acute in these kinds of searches.

Generalizing from these examples, we identify *interactive data exploration (IDE)* as an important class of human-centric search activity on images in which hypothesis formation and hypothesis validation proceed hand in hand in a tightly-coupled and iterative sequence. A user constructs an initial search predicate, gets back a few results, aborts the current search, and then modifies the search predicate (sometimes extensively) in the light of these results. This iterative process continues until the user finds what she is looking for, or gives up. As illustrated in Figure 5, the user is effectively conducting two interleaved and tightly-coupled searches: one on the query space (the space of all possible combinations of search predicates) and the other on the data space (all images). This interleaved workflow is consistent with the metaphor that asking exactly the right question about complex data is often the key to a major insight. However, the path to converging on that precise question may be long and convoluted with many false turns and dead ends. This workflow is the essence of IDE. If successful, you end with a search query that can be used as the basis of future classic indexing to rapidly answer similar queries — e.g., if this specific dog is ever lost again, the global catalog in Figure 1 will contain an index term to rapidly locate it.

Classic indexing, such as GigaSight's implementation from Section III, is *context-free.* The index is created in advance of use, without any knowledge that is only available at the time of a future search. In contrast, IDE is inherently *context sensitive.* The ability to deeply incorporate context-sensitive information into the search iterations of IDE is crucial to success.

V. Just-in-Time Indexing

For a human-in-the-loop system, the most precious resource is *user attention*. For IDE, we define a user's attention as being used well if most of it is spent on (a) examining individual results (i.e., video frames) to decide if they are true positives

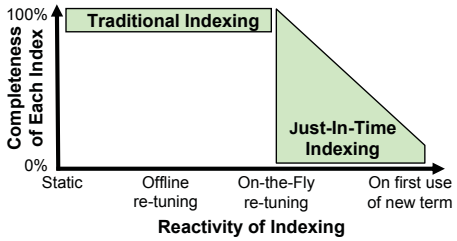

Fig. 6. Spectrum of Indexing Strategies

or false positives, or (b) on thinking about how the predicates of the current query should be modified for the next iteration of the IDE. We define user attention as being used poorly if most of it is spent on (a) waiting for the system to return results to examine, or (b) dismissing frivolous false positives from a search with low selectivity. Time is of the essence in IDE. In the working example of Figure 4, the owner's sole focus is finding her lost dog as soon as possible. She would prefer success sooner rather than later, even at the cost of more effort from her in IDE. This precludes approaches that defer IDE for many hours while the system performs background optimizations such as indexing or data restructuring.

To support IDE, we propose *just-in-time indexing (JITI)*, a new indexing strategy that exploits the following three properties of iterative query refinement:

- *Temporal locality of search predicates:* there is considerable redundancy in the queries posed during an IDE session. Each query is a refinement on the previous queries, often repeating many of the search predicates.
- *Rapid Refinement:* a user typically refines a query after seeing only tens of results. "Abort search" is frequent.
- *Considerable think time:* because IDE requires significant user reflection, there is an opportunity during think times to perform indexing.

As Figure 6 illustrates, JITI differs from previous approaches to indexing in two important ways. First, it is highly reactive to the current query session, building new indexes (or augmenting existing indexes) speculatively, on-the-fly during user think time. JITI exploits temporal locality of search predicates in the successive iterations of an IDE session by indexing any new search predicate on its first use. Second, JITI indexes only a small, adaptive subset of the images, instead of building complete indexes. This is sufficient because a user typically refines the content-based search query after seeing only tens of images returned. It is also necessary, given the prevalence of expensive predicates for image processing and the bounded amount of user think time (typically tens of seconds). While both speculative indexing [7] and partial indexing [8]–[10] have been proposed previously for relational databases, this work is the first to combine the two and to apply them in the context of image search.

We have built a prototype implementation of JITI that allows us to flexibly explore its design tradeoffs. Our prototype leverages the concept of *early discard,* whose importance in interactive image search was first established by Huston et al [11]. The code for parameterized image search predicates

(called *filters*) can be combined using a directed flow graph (called a *filter configuration*) into a composite *searchlet* that defines a search query. Predefined filters exist for color, texture, human faces, and many other image primitives. These can be parameterized during an IDE by the user (e.g. by using a color or texture patch from a previous result). JITI works at the granularity of individual filters. Conceptually, all the filters in the searchlet are executed *de novo*. However, JITI ensures that previously computed results can be used whenever they are still valid (i.e., neither filter code nor filter parameters have changed). Hence, a searchlet that reuses many previous filters unmodified will benefit from JITI.

Although our implementation is not integrated with GigaSight, we expect such integration to be straightforward. Here, we describe the steps of a search as it would occur in an integrated implementation. The term "image" in this description refers to a video frame that has been deemed to be within scope at the start of an IDE, based on timestamps and index terms in the global catalog. Scope can be dynamically changed as an IDE progresses. At the start of an iteration, the searchlet defining the query is shipped from the user's search front-end to all the cloudlets involved. The search proceeds independently at each cloudlet, and results are streamed back to the user as soon as each is generated. The search terminates at all cloudlets as soon as the user aborts the search. By then, the user has likely seen enough to create an improved searchlet for the next iteration of the IDE session.

VI. JITI Policies

JITI is performed independently at each cloudlet, as a transparent side effect of the search process. In response to a search query (defined by a filter configuration), the user starts seeing results from all the cloudlets intermingled as they are streamed to her. Her display pauses when the screen is full, but processing on cloudlets and streaming of results can continue in the background. Buffered results are presented to the user as she advances to new screens. The order in which images are evaluated on a cloudlet is left unspecified, thus allowing flexibility in optimizing the storage layer. Before applying a filter to an image, the cloudlet first checks to see if the result is already available in the index. Early discard ensures that processing on the image terminates as soon as it is clear that no path to success is possible with the current filter configuration.

A. Policies Studied

JITI is a broad concept that allows a wide range of flexibility in its implementation. The design parameters include: when indexing is triggered, which images are chosen for indexing, which filters are used in the indexing, how long indexing is continued, and so on. JITI policies can also vary in the weight they assign to the current query versus overall query trends. We have studied the following policies:

1. Current Query Work-Ahead: User think time is applied solely to working ahead on the current query. This optimizes for the case that the user requests more screenfuls of images, but is less effective if the user aborts the query immediately.

$\mathcal{O}$	universe of data objects
K	number of possible filters
F_j	filter j, $j = 1, \ldots, K$
D	a database of objects from $\mathcal{O}$
N	number of objects in the database
O_i	object with ID i, $i = 1, \ldots, N$
Q	a query (a conjunction of filters)
ℓ	the number of filters in a given query
t_j	CPU time (in ms) to evaluate F_j on an object
p_j	probability an object in D passes F_j; $p_j > 0$
f_j	probability a session contains F_j
r	probability that a filter used in a session is re-used within the same session
d	time (in ms) to fetch an object from disk
s	number of objects in a screenful
ϵ	given an object ID, time (in ms) to check if the object has been indexed and retrieve its pass/fail outcome

Fig. 7. Notation Used in the Cost Model

2. Popularity-Based: Statistics of filter use over a time window are maintained, and user think time is used to index the most popular filter. If indexing proceeds to completion, the next most popular filter is selected, and so on. This scheme optimizes for future queries that use these popular filters, at the expense of current query performance.

3. Efficiency-Based: Policy 2 is extended to recognize that slower (i.e., computationally more expensive) filters are more valuable to index because that can reduce user wait time. The policy also separately recognizes that filters with low pass-rates are especially valuable for early discard. Inaccuracies in filter cost estimation and pass rate are challenges.

4. Dimension Switching: Policy 2 is refined to scope popularity to only those filters that are actually used in the current query. Non-popular filters are evaluated only as needed to resolve pass/fail outcome. This policy balances the weight of the current query versus overall query trends.

5. Self-Balancing: This combines the other policies. Between queries, Policy 3 is used to optimize for future queries. Once a query is submitted, it is favored as follows. First, Policy 1 is used until the number of images awaiting user attention exceeds a predefined threshold. Then, Policy 4 is used until the number of images awaiting user attention exceeds a second predefined threshold. From that point onwards, Policy 3 is used until a new query is received from the user.

A comparison of these policies shows that Policy 5 (Self-Balancing) is the best. We summarize the analysis in the next section. A more detailed analysis of JITI policies can be found in our technical report [12].

B. Analysis of Policies

Our analysis is based on a cost model whose notation is shown in Figure 7. There is a large universe $\mathcal{O}$ of data objects (images, in our case) and a large universe $\{F_1, \ldots, F_K\}$, of filters. Each filter is a binary function that takes an object from $\mathcal{O}$ as its single argument and returns either pass or fail. A given database D contains N objects, $O_1, \ldots, O_N$, from $\mathcal{O}$. Users

pose queries to D, where each query Q is the conjunction of a finite set of filters.

The number of possible filters K is large because they stand for a huge number of semantic concepts. For example, in the animal domain alone, there may be separate filters for each animal species of interest (dogs, cats, raccoons, etc.) and perhaps even individual breeds within a species. Further, some feature selectors (e.g., color-selector or texture-selector) can be used to define an extremely large number of distinct filters. For such selectors, the user takes one or more exemplar images, selects a region of interest within each such image, and then defines a filter looking for "similar" regions in other images, where similarity is defined based on a color histogram or a texture histogram. In our implementation there are $\approx 10^{64}$ distinct possible color/texture histograms.

Associated with each filter F_j are (1) its *execution time*: the (average) time t_j in milliseconds to evaluate F_j on an object, (2) its *pass-rate* or *selectivity*: the fraction p_j of objects in a database D that pass F_j, and (3) its *session frequency*: the fraction f_j of sessions that include F_j. In practice, the system knows only approximations of these metrics, which are estimated over time. In the absence of information regarding filter pass rate correlations, we make the reasonable assumption that an object passes F_j with a probability p_j that is independent of all other filters. The parameter r refers to the probability that a filter occurring in a query is re-used within the same IDE session. For the traces in our experiments in Section VII, $r \approx 0.5$. We define d to be the time in milliseconds to retrieve an object from disk, and s to be the number of objects in a screenful ($s = 6$ in our implementaton). Finally, ϵ is the time to check if an object has been indexed and, if so, retrieve its pass/fail outcome. Note that $\epsilon \ll d$.

Response time, which is our primary performance metric, can be greatly affected by the degree of correlation in the identity of objects that are indexed by different partial indexes. There are an exponential number of ways in which partial indexes may overlap. These myriad possibilities do arise in practice because of early discard in a setting where different queries may use different filter combinations. To make the analysis tractable, we focus on simplified scenarios where a partial index is either complete or empty with respect to generating a screenful of results for a given query. We refer to these cases as "effectively complete" and "effectively empty." The key insights are still revealed with this simplification.

Consider a query $Q = F_1 \wedge \cdots \wedge F_\ell$, when there are effectively complete indexes on F_1 through F_i, and only effectively empty indexes on F_{i+1} through F_ℓ (for some i, $0 \leq i \leq \ell$). Assume that the filters are evaluated on an object O in order, i.e., first F_1, next F_2 if O passes F_1, next F_3 if O passes both F_1 and F_2, and so on. Then the expected response time, T, for generating a screenful of s objects is:

$$T = \frac{s \cdot \epsilon}{p_1 \cdots p_\ell} + \cdots + \frac{s \cdot \epsilon}{p_i \cdots p_\ell} + \frac{s \cdot (d + t_{i+1})}{p_{i+1} \cdots p_\ell} + \frac{s \cdot t_{i+2}}{p_{i+2} \cdots p_\ell} + \cdots + \frac{s \cdot t_\ell}{p_\ell} \quad (1)$$

Intuitively, Equation 1 reveals that we expect to access $\frac{s}{p_1 \cdots p_\ell}$ objects in order to find s that satisfy Q. We apply F_1 to all these objects (at cost ϵ per object). Because of early discard, subsequent filters are applied to fewer and fewer objects. There are $\frac{s}{p_{i+1} \cdots p_\ell}$ objects that pass all indexed filters; these objects must be fetched from disk (at cost d). Finally, each non-indexed filter F_j, $j = i+1, \ldots, \ell$, costs t_j per object. Equation 1 suggests the following rules-of-thumb:

R1: To minimize response time, first retrieve the outcome of indexed filters in non-decreasing order of p_j and then evaluate non-indexed filters in non-decreasing order of $\frac{t_j}{1-p_j}$.

R2: The best indexes to have on hand for a query Q are for filters in Q that have low pass-rates and slow execution times. However, the most important aspect is that the indexed filter occurs in Q.

R3: Because the probability r of intra-session re-use is orders of magnitude higher than the probability f_j of inter-session reuse for all but the most popular filters, schemes that focus on filters in the current query (Current Query Work-Ahead and Dimension Switching) are good choices for minimizing response times: they are effective both when the user requests the next screenful and when she instead refines the query.

R4: As long as there remain unindexed highly popular filters, indexing them is worthwhile. Using $\frac{f_j}{p_j}(d + t_j)$ as the priority metric in the Efficiency-based scheme is a good choice for inter-session indexing (assuming we have good estimates for f_j, p_j and t_j).

Together, these four rules support Self-Balancing as the preferred policy among the five proposed alternatives.

C. JITI Data Structures

Implementing JITI requires attention to two important index design considerations: (a) what type of index to use, and (b) how to structure the index catalog. Our design choices in these areas are particularly efficient in time and space.

The query workload provides a natural choice for what type of index to use. Given that our queries are conjunctions of pass/fail filters, we use bitmapped indexes for each filter. Note that the distinct values in our bitmapped index are *pass*, *fail* and *unknown*. The *unknown* value is necessary because our indexes are partial: at any point in time, we have evaluated the filter on only a subset of the images.

We build a bitmapped index on the values *pass* and *fail*: i.e., an index having a bitmap for which images pass and a bitmap for which images fail. Images that are *unknown* are hence represented only implicitly by their omission from the other two bitmaps. In this way, inserting new images into a database incurs no maintenance overhead: such images are implicitly tagged as *unknown* with respect to all existing indexes. This is in sharp contrast to traditional settings, where bitmapped indexes must be kept up-to-date, incurring high overheads.

Because a typical image is 100KB–10MB, the size of each (partial) bitmapped index for a filter is 6–8 orders of magnitude

	Description	Images	Hits
S1	Find all images of a specific person.	2582	8
S2	Find five instances of theft in a surveillance image dataset.	1072	6
S3	Find five pictures of sailboats or windsurfers.	281,324	476
S4	Find three pictures of urban outdoor scenes.	281,324	18,805
S5	Find ten pictures from a colleague's wedding.	281,324	67

Fig. 8. Search Tasks Emulating IDE Sessions

smaller than the total size of all the images that have been processed with that filter. This, in turn, implies that even a large number of indexes can be supported with very fast access times. Fetching an index is orders of magnitude faster than fetching the images. Once fetched, an index can reside high in the memory hierarchy. Moreover, their small space and lack of maintenance costs means that we can be lazy about discarding old indexes.

The catalog of all existing indexes is stored as an inverted index. Because of the large number of potential filters, many of which are ad hoc, we use the hash of the filter description as the key for the inverted index. By definition, the filter description completely characterizes the filter. For example, it may be (i) the name and version number of a predefined filter template together with any arguments, (ii) the name, version number, histogram, and other arguments for a user-defined color-selector or texture-selector filter, or (iii) the code binary for a custom code filter.

Together, these design choices make it possible to support up to tens of thousands of partial indexes, with very fast access times (orders of magnitude faster than accessing the images), low space overhead ($< 1\%$ overhead for all the indexes together) and no required maintenance costs.

VII. Experimental Results

The ideal experimental approach to comparing JITI policies would be to use benchmarks that capture real-world IDE workloads. Unfortunately, we are at a very early stage of the evolution of edge computing. It will be many years before cloudlets and GigaSight are widely deployed, and generate real-world IDE workloads. In the interim, we emulate IDE-like workloads by capturing and replaying the actions of real users in performing search tasks on static image collections.

The hardware setup for our experiments consists of four servers playing the role of cloudlets, connected to a client via a 1 Gbps Ethernet switch. All machines have 2.83 GHz Intel® Xeon® processors with 8 GB RAM, and run Ubuntu Linux. The *trace replay* approach mentioned above reproduces captured user workloads with realism and replicability, while providing tight control of experimental conditions.

	S1	S2	S3	S4	S5
User 1	7	7	7	2	6
User 2	6	3	4	2	9
User 3	3	14	4	3	4
User 4	3	3	6	3	2
User 5	3	7	5	4	5
User 6	16	11	5	1	11
User 7	10	3	4	5	2
User 8	14	22	5	1	12

Fig. 9. Queries per IDE Session

We captured traces of eight users on the five different search tasks summarized in Figure 8. Some of these tasks are vague by nature, and have many degrees of freedom. S1 and S2 work on relatively small collections of images and emphasize recall (fraction of relevant images retrieved against the total number of relevant images in the database). S3–S5 work on a much larger image collection and favor precision (fraction of relevant images retrieved against the number of retrieved images). The image repository employed for searches S3–S5 consists of 32,757 manually ground-truthed personal photographs augmented with 248,567 images downloaded from Flickr. A subset of 4,323 randomly-sampled images from the latter was manually labeled to estimate the number of matches in the Flickr collection.

A. Observed Query Attributes

Figure 9 shows the number of queries in IDE sessions. The average of six queries suggests that the IDE process is indeed iterative. Figure 10 shows the extent to which queries within an IDE session reuse filters. Repeated use of any filter within a session constitutes intra-session reuse. The first use of either a predefined color filter or a popular filter is shown as inter-session reuse. Subsequent uses of such filters are considered intra-session reuse. The amount and type of reuse determines the extent to which JITI can be successful. Filters used across sessions may be indexed by the Popularity-Based and Dimension Switching schemes during idle periods between and during sessions, respectively. Even filters unique to a session, if reused, benefit from the indexes created by Current Query Work-Ahead. The Self-Balancing scheme inherits the benefits of each of these constituent schemes.

The figure shows that reuse occurs in all but six of the sessions, and that most sessions exhibit both types of reuse.

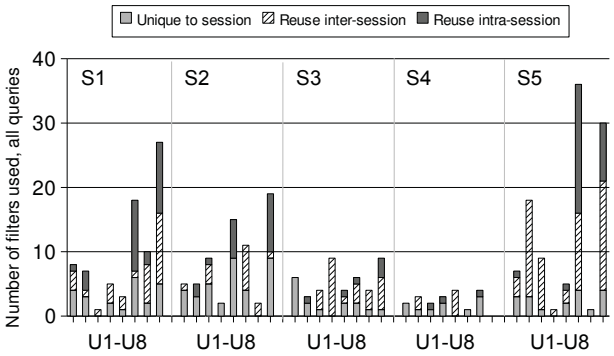

Fig. 10. Filter Reuse

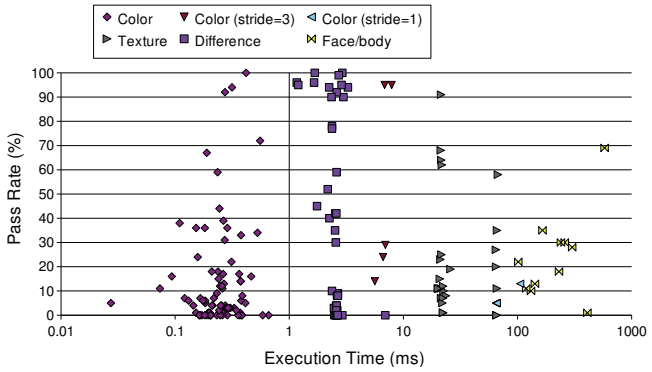

Fig. 11. Filter Execution Time and Selectivity (Log Scale on X Axis)

Of the total number of filters used, 70% are repeats. Nearly half of all filters defined are reused. The vast majority of user-defined filters are applied to a single session. Predefined filters tend to be used across sessions.

Queries are refined well before completion. Only 16% of the queries actually run to completion. For the recall searches, which are most likely to be exhaustive, 42% of the queries run to completion. For the precision searches, 7% of the queries run to completion. Users are able to evaluate their queries quickly and devise methods for refining them based on a small number of returned results. On average, users refine their queries after viewing only 36 images, and the queries process less than 10% of the images in the repository. Users exhibit considerable think time in performing the searches. Session length varies from less than 30 seconds (for U6 performing S4) to nearly 20 minutes (for U3 performing S5). 97% of the total search session time is think time. These periods of think time provide ample opportunities for JITI.

Figure 11 shows the speed and selectivity of the filters that are used. Over half of the defined filters have pass rates of 10% or less, and nearly one-fifth of the filters have pass rates of 1% or less. Average filter execution time, shown on a log scale, varies over three orders of magnitude depending on filter type. The most expensive filters are those for face and body detection from the OpenCV library.

B. Effectiveness of JITI

Using trace replay, we compare the performance of JITI to three other indexing schemes:

- **No indexing:** This is the worst case scenario.
- **Clairvoyant:** This is the ideal case, where indexes already exist for all the filters and images needed.
- **Workload-based:** We allot a budget of 100 milliseconds of CPU time per image. We index as many of our workload's most popular filters as we can within this budget. In addition, current query workahead is enabled.
- **JITI:** Each workload begins with global knowledge of filter use frequencies across all users. Only this user and workload are new to JITI. In addition, no indexes exist for any previously executed filters. This pessimistic restriction eliminates any benefit that could be realized from popularity-based indexing prior to the session.

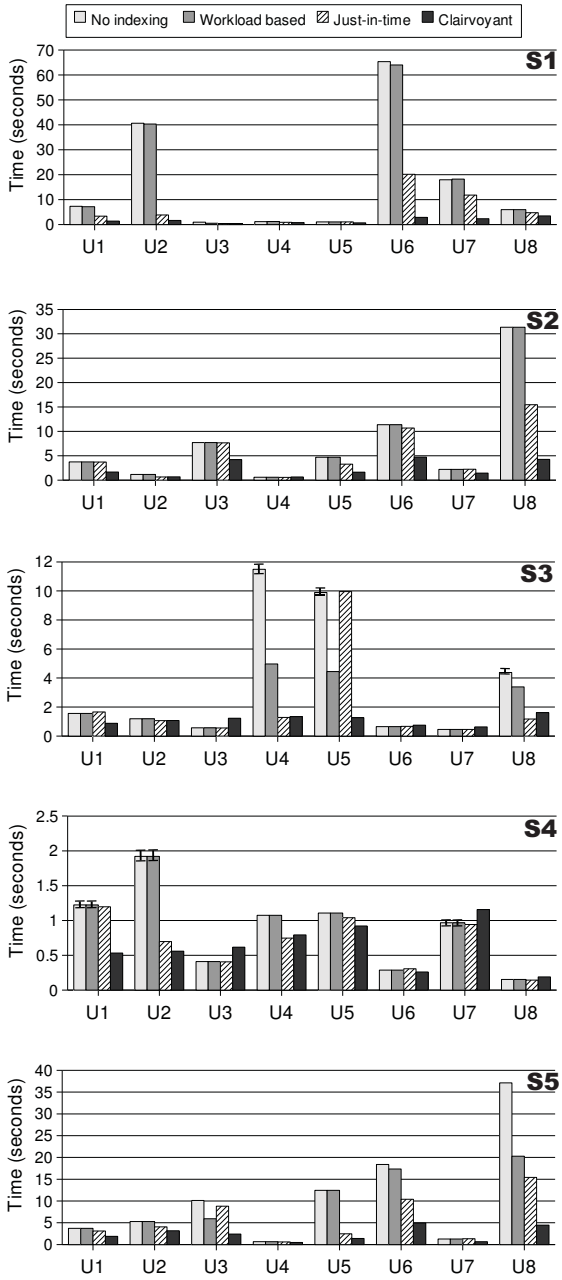

Fig. 12. Response Times (S1–S5) for Different Indexing Schemes

From the user's perspective, the primary figure of merit in IDE is *response time,* which is defined as the average time over the course of an IDE session that the user waits to receive a screenful of results after issuing a query, or requesting the next screenful of results. Across users and search tasks, Figure 12 compares response times across the four alternative indexing schemes. Each bar represents the average of three runs. The standard deviation is shown with error bars where it is large enough to be visible. The main message of these results is that *JITI offers significant benefit in many cases, and often matches or exceeds the performance of workload-based indexing.*

Figure 13 plots the data points in Figure 12 sorted by session response time for the no-indexing scheme. For half of the searches, the response time with no indexing is under

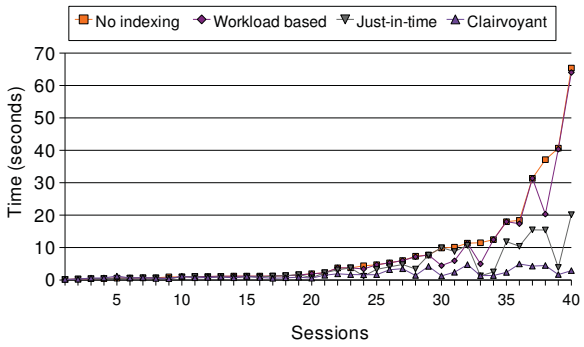

Fig. 13. Response Time Sorted by No-Indexing Time

2 seconds, so there is little room to improve response time. For the remaining searches, just-in-time indexing improved the response time by an average of 38% over no indexing, and 25% over workload-based indexing. The improvement was the largest in sessions with high filter reuse, and in particular, reuse of fast, selective filters. Such filters may be indexed extensively within a few queries, and low selectivity enables large numbers of images to be discarded rapidly.

VIII. Conclusion

The challenges of Internet-scale video capture, storage and use will become more acute over time, as video cameras proliferate and their resolution improves. Edge-based computing on cloudlets can alleviate this pain. By avoiding blind transmission of captured video to the cloud, cloudlets improve scalability by lowering ingress bandwidth demand. Only a tiny fraction of the captured video, selected for their importance and/or popularity, needs to transmitted to the cloud. By providing ample storage at the edge for extended retention periods of tens of days, cloudlets provide the opportunity for users to retrospectively discover important information that is buried in the captured video. These discoveries can have significant personal, business, and societal benefits.

Classic indexing on cloudlets using well-known computer vision algorithms is necessary, but not sufficient, to support the process of discovery from captured video. The final phase of this process is almost always context-sensitive, and requires incorporation of crucial information whose significance was not known at the time of index creation. In this paper, we have described a human-centric, interactive search process (IDE) for this final phase of discovery that leverages system support for early discard of image data at cloudlets. We have shown that the user think time and temporal locality inherent in IDE can be leveraged to perform partial and incremental indexing (JITI) for context-sensitive attributes. Our experiments confirm that JITI can improve interactive performance during IDE.

Acknowledgements

This research was supported in part by the National Science Foundation (NSF) under grant number CNS-1518865, and by the Defense Advanced Research Projects Agency (DARPA) under Contract No. HR001117C0051. Additional support was provided by Intel, Google, Vodafone, Deutsche Telekom, Verizon, Crown Castle, and the Conklin Kistler family fund. Any opinions, findings, conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the view(s) of their employers or the above-mentioned funding sources.

References

[1] D. Barrett, "One surveillance camera for every 11 people in Britain, says CCTV survey," *Daily Telegraph*, July 10, 2013.

[2] D. Lavrinc, "Why Almost Everyone in Russia Has a DashCam," *Wired*, February 15, 2013.

[3] S. Banerjee and D. O. Wu, "Final report from the NSF Workshop on Future Directions in Wireless Networking," NSF, November 2013.

[4] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies, "The Case for VM-Based Cloudlets in Mobile Computing," *IEEE Pervasive Computing*, vol. 8, no. 4, 2009.

[5] J. Shotton, M. Johnson, and R. Cipolla, "Semantic texton forests for image categorization and segmentation," in *Proc. of IEEE Conf. on Computer Vision and Pattern Recognition*, June 2008, pp. 1–8.

[6] P. Simoens, Y. Xiao, P. Pillai, Z. Chen, K. Ha, and M. Satyanarayanan, "Scalable Crowd-Sourcing of Video from Mobile Devices," in *Proc. of ACM MobiSys 2013*, 2013.

[7] N. Polyzotis and Y. Ioannidis, "Speculative Query Processing," in *Proceedings of CIDR Conference*, January 2003.

[8] C. Sartori and M. R. Scalas, "Partial Indexing for Nonuniform Data Distributions in Relational DBMS's," *IEEE Transactions on Knowledge and Data Engineering*, vol. 6, no. 3, June 1994.

[9] P. Seshadri and A. Swami, "Generalized Partial Indexes," in *Proceedings of IEEE ICDE*, March 1995.

[10] M. Stonebraker, "The Case for Partial Indexes," *ACM SIGMOD Record*, vol. 18, no. 4, December 1989.

[11] L. Huston, R. Sukthankar, R. Wickremesinghe, M. Satyanarayanan, G. Ganger, E. Riedel, and A. Ailamaki, "Diamond: A Storage Architecture for Early Discard in Interactive Search," in *Proceedings of the 3rd USENIX Conference on File and Storage Technologies*, San Francisco, CA, April 2004.

[12] P. B. Gibbons, L. Mummert, R. Sukthankar, M. Satyanarayanan, and L. Huston, "Just-In-Time Indexing for Interactive Data Exploration," School of Computer Science, Carnegie Mellon University, Tech. Rep. CMU-CS-07-120, April 2007.