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 in the Internet of Things, the ability to scalably capture and search that data becomes important. Scalability can be achieved by performing video analytics on cloudlets at the edge of the Internet, and only shipping extracted index information and meta-data to the cloud. This paper describes a technique called *just-in-time indexing* that extends such an architecture to support interactive content-based retrospective search of cloudlet data for predicates that were not part of the original indexing strategy.

I. Ubiquitous Video Cameras

Video cameras that are always on or frequently on 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]. By 2012, virtually every automobile in Russia had a video camera on its dashboard to record incidents for insurance purposes [2], [3]. Body-worn cameras are increasingly common in police forces [4]. Extrapolating from these trends, the report of the 2013 *NSF Workshop on Future Directions in Wireless Networking* [5] 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.

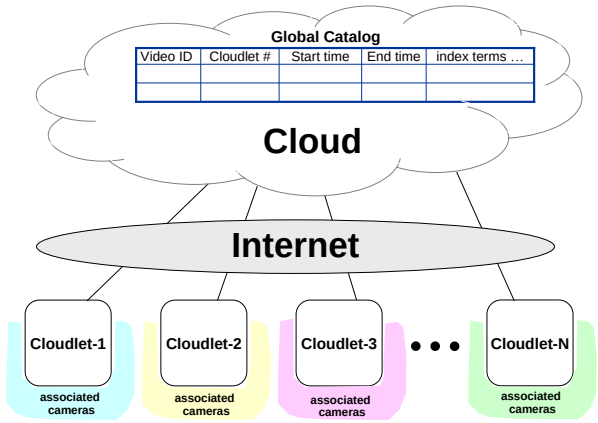

Fig. 1. Two-level Cloud-Cloudlet Architecture

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* [6]. 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

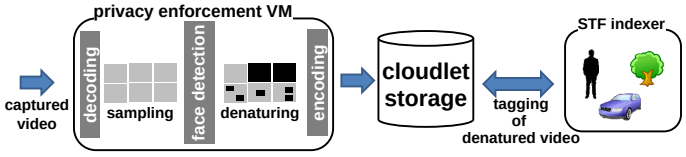

Fig. 2. GigaSight Video Processing Workflow on a Cloudlet

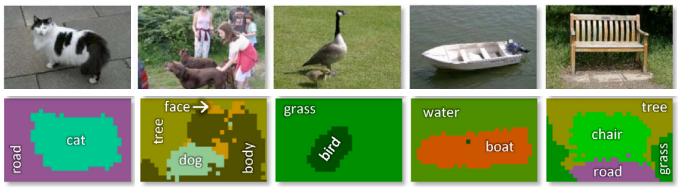

Fig. 3. Examples of STF Object Detection (Adapted from Shotton et al [7])

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

previous paper [8], 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 [7], 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 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

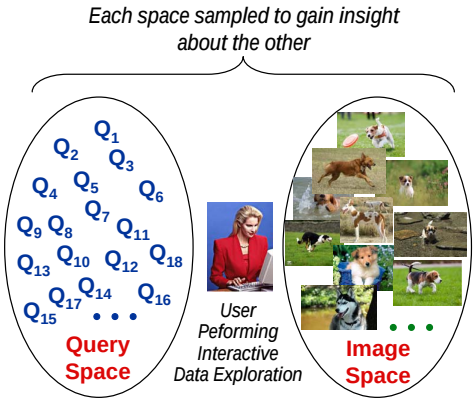

Fig. 5. IDE: Interleaved Search in Two Spaces

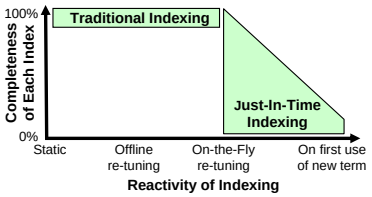

Fig. 6. Spectrum of Indexing Strategies

	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. 7. Queries per IDE Session

	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

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 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. We describe our solution in the next section.

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 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 [9] and partial indexing [10]–[12] 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 [13]. 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 yet 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. The early discard strategy mentioned earlier ensures that processing on the image can be terminated as soon as it is clear that no path to success is possible with the current filter configuration.

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. The range of policies include the following:

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.

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 is a combination of 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.

An analytical comparison of these policies (omitted here for space reasons) shows that Policy 5 (Self-Balancing) is the best. The details can be found in our technical report [14].

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

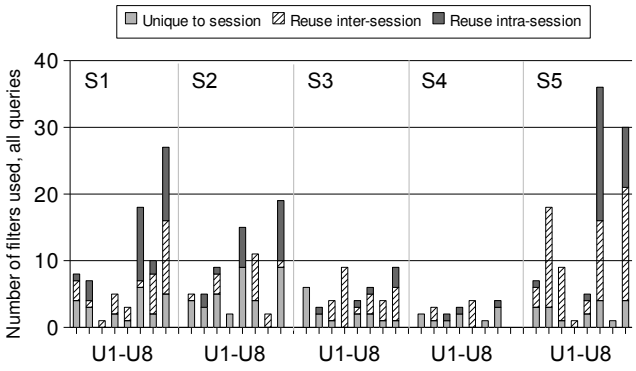

Fig. 9. Filter Reuse

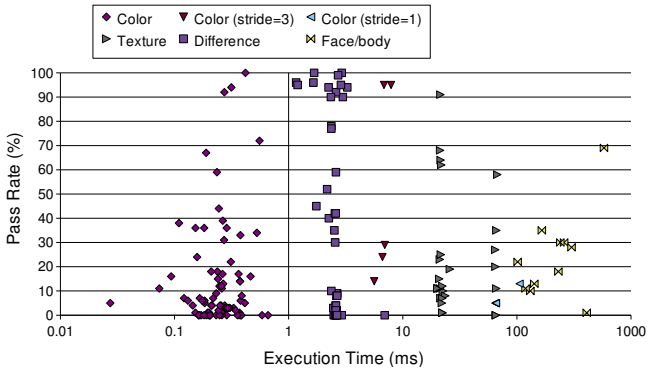

Fig. 10. Filter Execution Time and Selectivity (Log Scale on X Axis)

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.

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 7 shows the number of queries in IDE sessions. The average of six queries suggests that the IDE process is indeed iterative. Figure 9 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. 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 10 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.

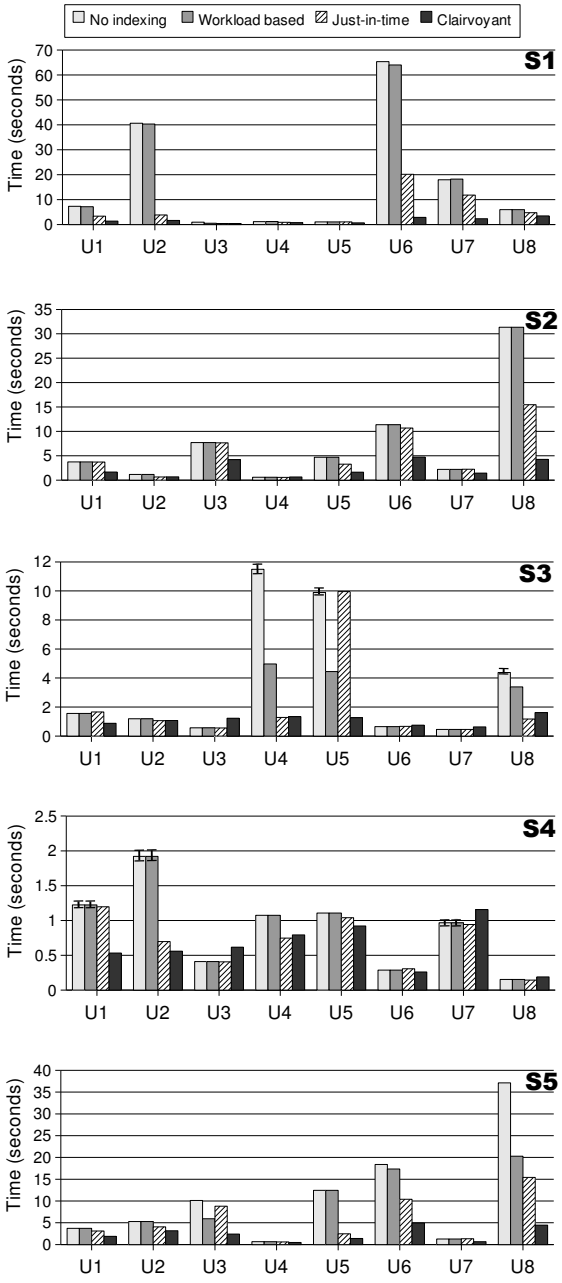

Fig. 11. Response Times (S1–S5) for Different Indexing Schemes

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 11 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.*

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.

References

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

[2] A. Davies, "Here's Why So Many Crazy Russian Car Crashes Are Caught On Camera," *Business Insider*, December 2012.

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

[4] L. Miller, J. Toliver, and Police Executive Research Forum 2014, *Implementing a Body-Worn Camera Program: Recommendations and Lessons Learned*. Washington, DC: Office of Community Oriented Policing Services, 2014.

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

[6] 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.

[7] 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.

[8] 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.

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

[10] 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.

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

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

[13] 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.

[14] 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.