FAST: Near Real-time Searchable Data Analytics for the Cloud

Yu Hua
Wuhan National Lab for Optoelectronics, School of Computer
Huazhong University of Science and Technology
Wuhan, China
csyhua@hust.edu.cn

Hong Jiang
Depart. Computer Science and Engineering
University of Nebraska-Lincoln
Lincoln, NE, USA
jiang@cse.unl.edu

Dan Feng
WNLO, School of Computer
Huazhong Univ. of Sci. and Tech.
Wuhan, China
dfeng@hust.edu.cn

***Abstract*—With the explosive growth in data volume and complexity and the increasing need for highly efficient searchable data analytics, existing cloud storage systems have largely failed to offer an adequate capability for real-time data analytics. Since the true value or worth of data heavily depends on how efficiently data analytics can be carried out on the data in (near-) real-time, large fractions of data end up with their values being lost or significantly reduced due to the data staleness. To address this problem, we propose a near-real-time and cost-effective searchable data analytics methodology, called FAST. The idea behind FAST is to explore and exploit the semantic correlation within and among datasets via correlation-aware hashing and manageable flat-structured addressing to significantly reduce the processing latency, while incurring acceptably small loss of data-search accuracy. The near-real-time property of FAST enables rapid identification of correlated files and the significant narrowing of the scope of data to be processed. FAST supports several types of data analytics, which can be implemented in existing searchable storage systems. We conduct a real-world use case in which children reported missing in an extremely crowded environment (e.g., a highly popular scenic spot on a peak tourist day) are identified in a timely fashion by analyzing 60 million images using FAST. Extensive experimental results demonstrate the efficiency and efficacy of FAST in the performance improvements and energy savings.**

***Keywords*—Cloud storage, data analytics, real-time performance, semantic correlation.**

I. Introduction

A cloud storage environment usually amasses huge volumes of data that critically require fast and accurate data retrieval to support intelligent and adaptive cloud services [1]–[3]. We have entered the era of the cloud characterized in part by the sheer volumes of data and the increasing reliance on the cloud storage. 7% of consumers stored their contents in the cloud in 2011, and the figure will grow to 36% in 2016, according to the Gartner, Inc. [4] and Storage Newsletter [5] reports. Average storage capacity per household will grow from 464 Gigabytes in 2011 to 3.3 Terabytes in 2016. So far, only a tiny fraction of the data being produced has been explored for their potential values through the use of data analytics tools. IDC estimates that by 2020, as much as 33% of all data will contain information that might be valuable if analyzed [6].

To parse and analyze a massive amount of files, conventional approaches mainly use content-based analysis tools that not only incur high complexity and costs but also fail to effectively handle these files. The high complexity routinely leads to very slow processing operations and very high and often unacceptable latency. Due to the unacceptable latency, the staleness of data severely diminishes the value of data. The *worth* or *value* of data in the context of data analytics means the valuable knowledge hidden in the data that can directly translate into economic values/gains in business-intelligence applications or new scientific discoveries in scientific applications. Since the value/worth of data typically diminishes with time, large amounts of data are often rendered useless, although costly resources, such as computation, storage and network bandwidth, have already been consumed to generate, collect and/or process these data. Therefore, we argue that (near-) real-time schemes are critical to obtaining valuable knowledge in searchable data analytics.

In the context of this paper, *searchable data analytics* are interpreted as obtaining data value/worth via queried results, such as finding a valuable record, a correlated process ID, an important image, a rebuild system log, etc. In the remainder of the paper, the term data analytics will be used to refer to searchable data analytics for brevity. In order to efficiently and effectively support (near-) real-time data analytics, we need to carefully address the following three research problems:

Unacceptable Access Latency. Existing approaches to unstructured data analytics rely on either system-based chunks of data files or multimedia-based features of images. The exact content-based methodology produces large amounts of auxiliary data (e.g., high-dimensional vectors, complex metadata, etc), which can be even larger than the original files. Even with the support of cloud platforms, it is non-trivial for these schemes to obtain the desired analytics results in a timely manner. For example, processing a typical image of 1MB, using the state-of-the-art PCA-SIFT approach [7], results in 200KB worth of features on average. This means that analyzing 1 million such images will lead to approximately 200GB of storage space requirement just for the features. A simple operation, such as finding a match for a given image from a 2-million-image set, would require 12.6 minutes of time on a commercial platform, due to frequent accesses to hard disks [8], [9].

High Query Costs. Data analytics for the cloud typically consume substantial system resources, such as memory space, I/O bandwidth, high-performance multicore processors (or GPUs). One of the main culprits for the high resource costs is the severe performance bottleneck frequently caused by query operations. In fact, many data-analytics related operations heavily rely on queries to identify the candidates for various operations. For example, query is the key process

978-1-4799-5500-8/14 $31.00 © 2014 IEEE
DOI 10.1109/SC.2014.67

for finding access patterns, correlated files, cost-effective data replication. Thus, we argue that improving query performance is of paramount importance to bridging the gap between data-analytics performance requirements and cloud system support. Moreover, massive images are generated by the smartphones of users who routinely take, share and upload pictures with their phone's HD cameras. These images collectively form huge data sets readily available for many data analytics applications. It's a known fact that users must charge their smartphones after a single day of moderate usage. In a 2011 market study conducted by ChangeWave [10] concerning smartphone dislikes, 38% of the respondents listed that the battery life was their biggest complaint, with other common criticisms such as poor 4G capacity and inadequate screen size lagging far behind. A substantial fraction of energy consumption in mobile phones may be caused, arguably, by frequently taking and sharing pictures via the cloud (uploading/downloading). An intuitive idea is to significantly reduce the number of images to be uploaded by sharing (and uploading) only the most representative one rather than all, at least when the mobile phone is energy-constrained. This idea is feasible since the images to be uploaded are often identical or very similar to the ones that have already been stored in the servers of the cloud. The challenge thus lies in how to efficiently and accurately identify such identical and similar images.

Diminished Analytics Values. Due to the long latency incurred in data processing and the resulting data staleness, the value/worth of data becomes diminished and eventually nullified. In some cases, the results of data analytics on stale data can even be misleading, leading to potential fatal faults. For instance, the prediction for earthquake, tsunami and tornado relies heavily on analyzing large amounts of data from earthquake sensors, ocean-mounted bottom sea-level sensors and satellite cloud imagery. The analysis must be completed within a very limited time interval to avoid or minimize disastrous results.

Real-time data analytics are very important in dealing with large-scale datasets. This is also non-trivial to cloud systems, although they contain high processing capability (hundreds of thousands of cores) and huge storage capacity (PB-level). The fundamental reason is because the analytics must be subject to hard time deadlines that usually cannot be met by brute force with an abundance of resources alone. Existing approaches often fail to meet the (near-) real-time requirements because they need to handle high-dimensional features and rely on high-complexity operations to capture the correlation.

To address the above problems facing real-time data analytics, we propose a novel near-real-time methodology for analyzing massive data, called FAST, with a design goal of efficiently processing such data in a real-time manner. The key idea behind FAST is to explore and exploit the correlation property within and among datasets via improved correlation-aware hashing [11] and flat-structured addressing [12] to significantly reduce the processing latency of parallel queries, while incurring acceptably small loss of accuracy. The approximate scheme for real-time performance has been widely recognized in system design [13]–[16] and high-end computing [17]–[19]. In essence, FAST goes beyond the simple combination of existing techniques to offer efficient data analytics via significantly increased processing speed. Through the study of the FAST methodology, we aim to make the following contributions for near real-time data analytics.

First, in order to efficiently and effectively identify similar files in a real-time manner, FAST makes use of a Bloom-filter based summarization representation that is simple and easy to use, by hashing the large-size vectors of files into space-efficient Bloom filters. Two similar files generally contain multiple identical vectors. Bloom filters can maintain the memberships of vectors and succinctly represent the similarity of files. Due to the space efficiency, substantially more membership information can be placed in the main memory to significantly improve the overall performance.

Second, FAST significantly improves the energy efficiency in the smartphones that leverage a near-deduplication scheme to substantially reduce the amount of similar images to be transmitted. Our design alleviates the computation overheads of existing schemes for similarity detection of files by using Locality-Sensitive Hashing (LSH) [11] that has a complexity of $O(1)$ to identify and aggregate similar files into correlation-aware groups. This allows the retrieval to be narrowed to one or a limited number of groups by leveraging correlation awareness. Unlike conventional hashing schemes that try to avoid or alleviate hash collisions, LSH actually exploits the collisions in its vertical addressing to identify the potential correlation in a real-time manner.

Third, due to the variable lengths of linked lists, LSH hash tables will likely lead to unbalanced loads and unpredictable query performance of vertical addressing. To address this problem, FAST optimizes its LSH-based hash functions by means of a manageable flat-structured addressing scheme using a novel cuckoo-hashing based storage structure to support parallel queries. FAST exploits the semantic correlation to offer an $O(1)$ addressing performance.

Fourth, we implement all components and functionalities of FAST in a prototype system. The prototype system is used to evaluate a use case of near real-time data analytics of digital images. We collect a big and real image set that consists of more than 60 million images (over 200TB storage capacity) taken of a top tourist spot during a holiday. In the cloud, instantaneously uploading and widely sharing images are growing as a habit and a culture, which helps form large reservoirs of raw images on which accurate analytics results may be obtained. Using this real-world image dataset as a case study, we evaluate the performance of FAST of finding missing children from the image dataset and compare it with the state-of-the-art schemes. The case study evaluation demonstrates the efficiency and efficacy of FAST in the performance improvements and energy savings.

The rest of this paper is organized as follows. Section II presents the results of a user survey, as well as the FAST methodology. Section III describes the FAST architecture and implementation details. We evaluate the FAST performance in Section IV. Section V presents the related work. Section VI concludes the paper.

II. FAST METHODOLOGY

The idea behind FAST is to explore and exploit the semantic correlation property within and among datasets via

correlation-aware hashing [11] and flat-structured addressing [12] to significantly reduce the processing latency, while incurring acceptably small loss of accuracy.

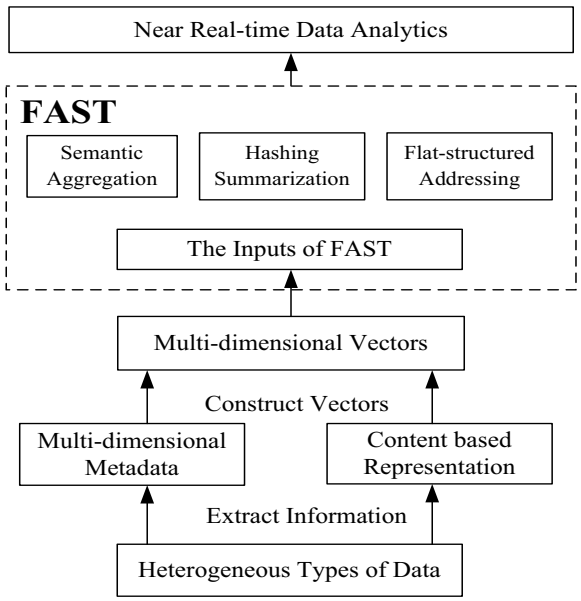


Fig. 1. The FAST methodology for multiple data types.

Semantic correlations measure the affinity among files. We use correlations to estimate the likelihood that all files in a given correlated group are of great interest to a user or to be accessed together within a short period of time [20], [21]. Affinity in the context of this research refers to the semantic correlation derived from multi-dimensional file attributes that include but are not limited to temporal or spatial locality. We derive this measure from multiple attributes of files, also called multi-dimensional correlation. To put things in perspective, linear brute-force search approaches use no correlation, which we call zero-dimensional correlation. Spatial-/temporal-locality approaches, such as Spyglass [22], SANE [23] and SmartStore [24], use limited-dimensional correlations either in access time or reference space, which can be a special case of our proposed approach. The main benefit of measuring semantic correlations in multi-dimensional attribute space is that the affinity among files can be more accurately identified.

A. Extension to Multiple Types of Data

The FAST methodology can be extended to and well suited for multiple data types. The generality of FAST can be explained as follows, along with Figure 1 and the two key phases of the FAST process. First, most data types can be represented as vectors based on their multi-dimensional attributes, including metadata (e.g., created time, size, filename/record-name, etc.) and contents (e.g., chunk fingerprints, image interest points, video frames, etc.). FAST extracts key property information of a given type in the form of multi-dimensional attributes and represents this information in multi-dimensional vectors (i.e., multi-dimensional tuples). Each dimension is one component of the vector. Second, the vector-based representation is fed as input to FAST for the subsequent operations of hash-based summarization, semantic aggregation and flat-structured addressing. In essence, the hash computation meets the needs of handling heterogeneous types of data. Hence, FAST as a methodology has the potential to efficiently support the analytics for heterogeneous types of data.

As shown in Table I, we elaborate on the corresponding relationship between the modules of the FAST methodology and typical searchable storage systems, such as Spyglass [22] and SmartStore [24], as well as a use case illustrated in Section II-B. The corresponding relationship includes the vector extraction (VE) for metadata and content, and the data analytics (DA) in a near real-time manner. The comparisons and analysis can be considered in two aspects. First, FAST is a generalizable methodology, of which some components and aspects are derived from and have been partially used in existing storage systems, such as Spyglass and SmartStore. However, due to their specific and custom designs, these systems, while achieving their original design goals, fail to efficiently support near real-time data analytics. Second, by incorporating the FAST methodology, existing systems can be enhanced to achieve better performance. For example, the LSH algorithm with $O(1)$ complexity and the cuckoo-driven storage of FAST can respectively accelerate semantic aggregation and provide flat-structured addressing for queries. We believe that FAST has the potential to be used in multiple storage systems with several data types.

B. A Use Case and Its Problem Statement

To implement FAST and examine the efficiency and efficacy of the proposed methodology, we leverage "*Finding Missing Children*" as a use case to elaborate the FAST design and evaluate its performance. A missing child is not only devastating to his/her family but also has negative societal consequences. Although existing surveillance systems are helpful, they often suffer from the extremely slow identification process and the heavy reliance on manual observations from overwhelming volumes of data.

There exists a large amount of similar multimedia images in the cloud (e.g., images of celebrities, popular sceneries, and events), as a result of people's habits, such as the tendency to take the pictures of the same scene multiple times to guarantee the quality of their images. Furthermore, many photo-sharing sites, such as Facebook, Flickr, and Picasa, maintain similar images from friends with common interests. Due to the wide existence and explosive growth of such duplicate and similar images, commercial sites, such as Imagery, Google, Yahoo!, Bing Images search, Picsearch, Ask Images Search, etc., have already begun to address this practical problem. It is of paramount importance to address the data-volume challenge facing data analytics in the cloud systems.

We propose to use a crowd-based aid, i.e., personal images that can be openly accessed, to identify helpful clues. People often take many similar pictures on a famous scenic spot, which actually are the snapshots of those locations in a given period of time. High-resolution cameras offer high image quality and multiple angles. Repeatedly taking pictures can further guarantee the quality of snapshots. Given the convenient and easy access to the cloud, these images are often uploaded and shared on the web instantaneously (e.g., by smartphones). We

TABLE I. THE RELATIONSHIP AND CORRESPONDENCE BETWEEN THE FAST METHODOLOGY AND EXAMPLE SYSTEM IMPLEMENTATIONS.

	FAST Methodology	**Use-case (Images)**	**Spyglass** [22]	**SmartStore** [24]
Data Analytics	Flat-structured Addressing	Cuckoo Hashing Storage	Hierarchical Addressing	Hierarchical Addressing
	Semantic Aggregation	LSH based Clustering	Subtree Partitioning	Latent Semantic Indexing
	Hash Summarization	Summary Vectors	Membership Bloom Filters	Membership Bloom Filters
Vector Extraction	Content Description	PCA-SIFT Features	Signature Files	No
	Metadata Representation	Vectors	K-D Tree	R-Tree

can therefore leverage these publicly accessible images made possible in part by the crowdsourcing activities to help find the images that are correlated with a given missing child. For example, if someone takes pictures in the Big Ben, the images possibly contain not only the intended men/women, but also occasionally other people, such as a missing child in the background. If this image is uploaded and open to the public (openly accessible), we have an opportunity to find the missing child based on the input of his/her image. We can quickly obtain the clues suggesting whether the missing child had ever appeared around the Big Ben. This clue helps us locate the missing child. The rationale comes from the observations that instantaneously uploading and widely sharing images are becoming a habit and culture in the cloud.

We must admit that the effectiveness of this approach is probabilistic. For instance, if this valuable image is not uploaded and not publicly accessible, FAST will fail to identify the clues, while consuming some system resources. However, based on our observations and real-world reports, users are becoming increasingly willing to share their sightseeing images due to the shared interests and the easy access to the Internet. In the meantime, our approach is orthogonal and complementary to existing surveillance systems in fast locating the missing children, by avoiding brute-force frame-by-frame manual checking upon massive monitor videos. In this way, only the correlated segments will be checked carefully to obtain significant time savings. By considering the incomparable value of finding missing children, the modest costs are obviously acceptable.

III. DESIGN AND IMPLEMENTATIONS

In this Section, we present the architecture and implementation details of the FAST methodology via a use case.

A. Architecture of Use Case

FAST supports a fast and cost-effective scheme for near real-time data analytics. It employs a simple and easy-to-use index structure with three unique properties: space-efficient summarized vectors, semantic-aware hashing and flat-structured addressing for queries. The summarized vectors fit the index into the main memory to improve indexing performance. The semantic-aware hashing significantly reduces the complexity of identifying similar images. The flat-structured addressing offers $O(1)$ complexity for real-time queries.

The proposed FAST methodology is implemented as a system middleware that can run on existing systems, including the Hadoop file system, by using the general file system interface and exploiting correlation property of data. Figure 2 shows the architecture of FAST in the use case of "Finding Missing Children". The correlation-awareness feature of FAST not only offers various services to users (e.g., queries), but also

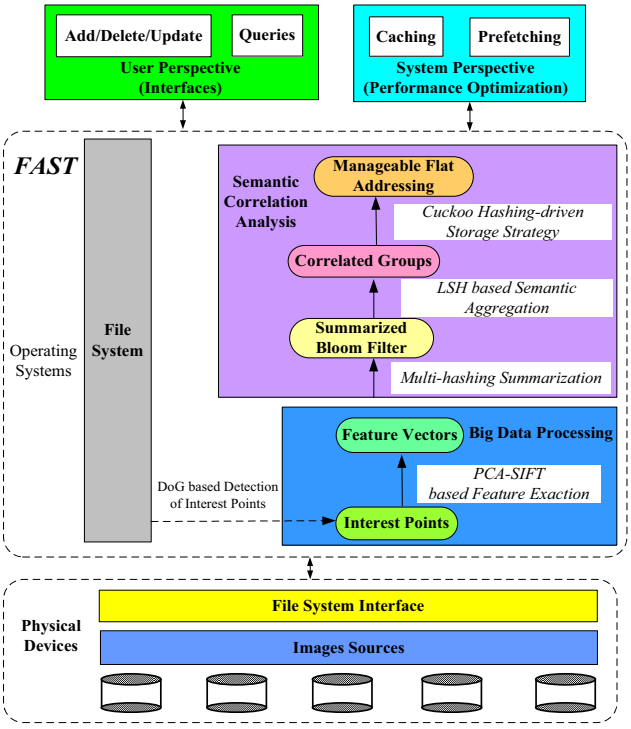


Fig. 2. The FAST implementation of the image-identification use case.

supports system optimization, such as caching and prefetching. FAST consists of two main functional modules, i.e., big data processing and semantic correlation analysis. Specifically, the former provides the function of Feature Extraction (FE) (i.e., lightweight feature extraction) based on the detection of interest points, while the latter consists of Summarization (SM) (i.e., space-efficient summarized vectors), Semantic Aggregation (SA) (i.e., semantic-aware grouping) and cuckoo hashing-driven storage (CHS) (i.e., manageable flat-structured addressing). The FE function makes use of the DoG [25] and PCA-SIFT schemes [7] to respectively detect and represent interest points of an image. In the computer vision field, an interest point refers to the point that is stable under local and global perturbations in the image domain. By capturing their interest points, FAST can identify and extract the features of similar images.

The identified features generally require a relatively large space for representation, for example, 200KB per 1MB image in the state-of-the-art PCA-SIFT scheme [7]. One billion such images would thus require about 200TB storage space. The storage and maintenance of these features consume substantial space, usually too large to be fully held in the main memory. The SM module, based on Bloom filter [26], is therefore

designed to represent these features in a more space-efficient manner. The Bloom filters in SM hash the input features into constant-scale positions in a bit array. Since only the hashed bits need to be maintained, these filters help significantly reduce the space requirement of features.

In the SA module, FAST employs locality sensitive hashing (LSH) [11], [27] to capture correlated features that identify similar images. In the CHS module, we make use of the cuckoo hashing structure to store the data items that incur hash collisions. The cuckoo hashing is essentially a multi-choice scheme to allow each item to have more than one available hashing position. The items can "move" among multiple positions to achieve load balance and guarantee constant-scale complexity of queries. However, a simple and naive use of cuckoo hashing in LSH will likely result in frequent operations of item replacement and potentially incur high probability of rehashing due to limited available buckets. This can lead to a severe performance bottleneck. We address this problem via adjacent neighboring storage as described in Section III-C3.

The above functional modules enable FAST to reduce the need for on-disk index lookups and decrease the complexity of identifying similar images. The workflow can be summarized as follows. First, the FE module is used to detect the interest points in the similar images with the DoG scheme and the detected interest points are represented by the PCA-SIFT scheme in a compact way to obtain substantial space savings. In the second step, in order to obtain further space savings and efficiently support semantic grouping, the SM module hashes the features per image into a space-efficient Bloom-filter based indexing structure. The rationale behind this strategy comes from the observations that similar images contain some identical features that project the same bits onto the Bloom filters. Therefore, the bit-aware Bloom filters can conjecture similar images. Finally, the Bloom filters are fed as inputs to LSH in the SA module. SA uses semantic-aware multiple hash functions to aggregate correlated images together. The correlated images are then stored in a cuckoo-hashing manner.

B. Features of Images

The *features of an image* are invariant to the scale and rotation of the image, thus providing robust matching across a substantial range of affine distortion, changes in various viewpoints, additions of noise, and changes in illumination. *Interest points* are effective local descriptions of image features and widely employed in real-world applications such as object recognition and image retrieval because they are robust to photometric changes and geometric variation and can be computed efficiently. Therefore, we use interest points in FAST to capture similarity properties of images.

To perform reliable and accurate matching between different views of an object or scene that characterize similar images, we extract distinctive invariant features from images. Feature-based management can be used to detect and represent similar images to support correlation-aware grouping and similarity search. Potential interest points are identified by scanning the image over location and scale. This is implemented efficiently by constructing a Gaussian pyramid and searching for local peaks in a series of difference-of-Gaussian (DoG) images.

We construct a local image descriptor for each interest point, based on the image gradients in its local neighborhood. The local image gradients are measured at the selected scale in the region around each interest point, and are transformed into a representation that allows for local shape distortion and change in illumination. Moreover, we apply principal components analysis to the normalized gradient patch. The patch covers an area in the original image that is proportional to the size of the interest point. The vector-based representation is both more distinctive and more compact, leading to significant improvements in matching accuracy and processing speed.

C. Semantic-aware Grouping

1) The Summary Vectors as Inputs: The feature-based representation generally requires large-sized memory. In order to reduce space overhead, we use Bloom-filter based bits as the input of semantic grouping to obtain significant space savings [26]. The space-efficient representation allows the main memory to contain more features. In general, two similar images imply that they contain many identical features. The identical features are hashed into the same bit locations in Bloom filters. Hence, two Bloom filters representing two similar images will share a significant number of identical bits. In the multi-dimensional space, each Bloom filter can be considered as a bit vector. Two similar Bloom filters can represent close-by items by virtue of their Hamming distance. Two similar images can be represented as two near-by points/items in the multi-dimensional space.

A Bloom filter is a bit array of m bits representing a dataset $S = \{a_1, a_2, \ldots, a_n\}$ of n items. All bits in the array are initially set to 0. A Bloom filter uses k independent hash functions to map items of the dataset to the bit vector $[1, \ldots, m]$. Each hash function maps an item a to one of the m-array bit positions. To determine whether an item a is an exact member of dataset S, we need to check whether all k hash-mapped bit positions of a are set to 1. Otherwise, a is not in the set S.

Bloom filters are used as the input to the locality sensitive hashing (LSH) module to fast and efficiently identify similar images. Since not all bits need be maintained, we only need to store the non-zero bits to reduce space overhead. For example, for a given image, the space required by its features can be reduced from the original 200KB to 40B, a 5000-fold space reduction, with only $O(1)$ computational complexity.

2) Semantic Grouping Scheme: To identify and group similar images, we leverage LSH to map similar images into the same hash buckets with a high probability [11]. Owing to its simplicity and ease of use, Bloom-filter based representation is used as LSH's input to reduce the complexity and accelerate the processing. Moreover, LSH function families have the locality-aware property, meaning that the images that are close to one another collide with a higher probability than images that are far apart. We define S to be the domain of images.

Definition 1: LSH function family, i.e., $\mathbb{H} = \{h : S \rightarrow U\}$, is called (R, cR, P_1, P_2)-sensitive for distance function $||*||$ if for any $p, q \in S$

- If $||p,q|| \leq R$ then $Pr_{\mathbb{H}}[h(p) = h(q)] \geq P_1$,
- If $||p,q|| > cR$ then $Pr_{\mathbb{H}}[h(p) = h(q)] \leq P_2$.

To allow similarity identification, we choose $c > 1$ and $P_1 > P_2$. In practice, we need to widen the gap between P_1 and P_2 by using multiple hash functions. Distance functions $||*||$ correspond to different LSH families of l_s norms based on an s-stable distribution to allow each hash function $h_{a,b} : R^d \rightarrow Z$ to map a d-dimensional vector v onto a set of integers. The hash function in $\mathbb{H}$ can be defined as $h_{a,b}(v) = \lfloor \frac{a \cdot v + b}{\omega} \rfloor$, where a is a d-dimensional random vector with chosen entries following an s-stable distribution and b is a real number chosen uniformly from the range $[0, \omega)$, where ω is a constant.

Each image representation consists of Bloom-filter based vectors, which are the inputs to LSH grouping mechanism. LSH computes their hashed values and locates them in the buckets. Since LSH is locality-aware, similar vectors will be placed into the same or adjacent buckets with a high probability. We select them from the hashed buckets to form the correlation-aware groups and support similarity retrieval.

Due to the property of hash collisions, which is exploited to identify similar images, LSH may introduce false positives and false negatives. A false positive means that dissimilar images are placed into the same bucket. A false negative means that similar images are placed into different buckets. In general, false negatives may decrease query accuracy and false positives may increase system computation and space overheads. Since reducing false negatives increases query accuracy and thus is more important than reducing false positives, we leverage extra probes by grouping not only the same, but also the adjacent buckets into a group. This is based on the locality-ware property of LSH, meaning that close-by buckets have stronger semantic correlation than far-apart ones. This methodology has been well verified by multi-probe LSH [28].

3) Flat-structured Addressing: Conventional LSH is able to group locality-aware data via exploring and exploiting the correlation in multi-dimensional attributes. In practice, this LSH scheme needs to alleviate high time and space overheads from vertical addressing. The vertical addressing is interpreted as the linear retrieval in a linked list that is generally used to avoid or mitigate hash collisions. However, due to no strict latency bounds of carrying out the vertical addressing, existing systems fail to obtain real-time query performance. In order to offer real-time performance in the cloud, we leverage flat addressing that executes cuckoo-hashing based operations and only incurs $O(1)$ complexity [12]. The cuckoo hashing based approach, in essence, exhibits query parallelism that can in turn be easily exploited by modern multi-core processors for performance improvements.

The flat-structured addressing probes a constant-scale number of buckets in parallel, each of which maintains one data item to offer $O(1)$ complexity, rather than checking the nondeterministic-length linked lists in conventional hash tables. The name of cuckoo-hashing method was inspired by how cuckoo birds construct their nests. The cuckoo birds recursively kick other eggs or birds out of their nests [12], [29]. This behavior is akin to hashing schemes that recursively kick items out of their positions as needed. The cuckoo hashing uses two or more hash functions to alleviate hash collisions and in the meantime decrease the complexity of querying the linked lists in the conventional hash tables. A conventional hash table generally provides a single position for placing an item a. The cuckoo hashing can offer two possible positions, i.e., $h_1(a)$ and $h_2(a)$, thus significantly alleviating the potential hash collisions and supporting flat addressing.

IV. Performance Evaluation

In order to evaluate the performance of the proposed FAST methodology for near real-time data analytics, we use a case-in-point scenario. This application aims to identify images similar to a given set of portraits from large image datasets in the cloud. A potential use case of this application could be to find a child reported missing in a crowded park by identifying images containing features similar to the given portraits of this child (e.g., by his/her parents) from images taken and uploaded by tourists of that park in the past few hours. The rationale for this is threefold. First, this application has the strong requirements for near real-time processing, for which long query latency will severely weaken the value/worth of the results. Second, to offer fast query performance, an efficient data structure, rather than a simple index structure is required for the large image store to facilitate semantic grouping and narrow the query scope. Third, due to the post-verification property, e.g., results will be verified by the missing child's parents or guardians, this use case is tolerant to small false results, which trades for significantly increased query efficiency.

A. Experiment Setup

We implemented a FAST prototype of the use case on a 256-node cluster. Each node has a 32-core CPU, with a 64GB RAM, a 1,000GB 7200RPM hard disk and a Gigabit network interface card. The implementation required approximately 1200 lines of C code in the Linux environment. To drive the FAST prototype evaluation, we use a real and large image dataset collected from the cloud. Initially, the image dataset is randomly distributed among the nodes. FAST then uses space-efficient and correlation-aware techniques for fast and efficient image indexing.

1) Evaluation Workload: Real Image Dataset: We collect real and openly assessable images from the popular campus networks of multiple universities, in the Cities of Wuhan and Shanghai in China, and well-known social networks. In order to faithfully demonstrate the real-time property of real-world image datasets, we set certain temporal and spatial constraints on the collection. First, the temporal constraint defines the uploading interval to be between a week-long holiday. This temporal constraint may potentially introduce some false positives and false negatives. For example, some images uploaded within this interval may actually record the contents (e.g., events/scenes/tourists) of a time or times prior to the interval, which leads to the false positives. We observe the percentage of false-positive images to be around 9.7% by using post-processing image tools and studying users' comments. These false-positive images are then filtered out. In fact, the false-positive images are tolerable in that they can be easily identified by users in the final query results. Furthermore, given the fact that more and more users are willing to share the images in a real-time manner, we believe that the percentage of false-positive images will gradually shrink in the future.

On the other hand, some images do record the contents of the defined time interval, but were either never uploaded

or uploaded after the defined interval, which results in the false negatives. Although it is non-trivial, if not impossible, to accurately calculate the percentage of such false-negative images, we attempt to approximately estimate this percentage by extending the upload interval by 3 days. About 16.8% false-negative images were uploaded in this 3-day interval. For hard real-time tasks (like finding missing children within 1 hour), these images are actually not accessible right away and only become helpful for post-analysis. Given the increasingly easier access to the Internet and the increasing tendency for users to share their pictures of the scenes almost on the spot and thus real-time, we argue that the percentage of false-negative images will also decrease.

The spatial constraint confines the locations to Wuhan and Shanghai in China, with each having its own unique and popular landmarks and sceneries. While Wuhan has 16 such landmarks, Shanghai has 22. We only collect images that contain these representative landmarks, which facilitates a meaningful evaluation. The collected image dataset ultimately contains 60 million images that amount to more than 200TB in storage size. The key characteristics of the image dataset are summarized in Table II. Moreover, the query requests, which are simultaneously issued from 500 clients, consist of the queried portraits in the real datasets.

TABLE II. THE PROPERTIES OF COLLECTED IMAGE SETS.

Datasets	No. Images	Total Size	File Type	Landmarks
Wuhan	21 million	62.7 TB	bmp(11%), jpeg(74%), gif(15%)	16
Shanghai	39 million	152.5 TB	bmp(9%), jpeg(79%), gif(12%)	22

2) Evaluation Baselines, Metrics and Parameters: We compare FAST with the state-of-the-art schemes, SIFT [30], PCA-SIFT [7] and real-time near-duplicate photo elimination (RNPE) [9]. Since there are no complete open-source codes, we choose to re-implement them. PCA-SIFT is a popular and well-recognized feature extraction approach that uses principal components analysis (PCA) for dimensionality reduction to obtain compact feature vectors. We implement scale-invariant feature transform (SIFT) [30], principal components analysis, point matching, query interface and storage tools. Moreover, RNPE studies the features of different location views to carry out real-time photo elimination. We implement its location visualization framework to retrieve and present diverse views captured within a local proximity.

The performance of FAST is associated with its parameter settings. One of the key parameters is the metric R that regulates the measure of approximate membership. The LSH-based structures can work well if R is roughly equal to the distance between the queried point q and its nearest neighbors. Unfortunately, identifying an optimal R value is a non-trivial task due to the uncertainties and probabilistic properties of LSH [11], [31]. In order to obtain appropriate R values for our experiments, we use the popular and well-recognized sampling method that was proposed in the original LSH study [32] and has been used in practical applications [27], [33]. We define "*proximity measure* $\chi = ||p_1^\star - q||/||p_1 - q||$" to evaluate the top-1 query quality for queried point q, where $p_1^\star$ and p_1 respectively represent the actual and searched nearest neighbors of point q by computing their distances. We determine the suitable R values to be 600 and 900 respectively for the *Wuhan* and *Shanghai* image datasets to appropriately and quantitatively represent the correlation. In addition, to construct the indexing structures, we use $L = 7$, $\omega = 0.85$, $M = 10$ in the LSH-based computation and $k = 8$ for the hash functions in the Bloom filters based on the above sampling mechanism.

The accuracy of approximate queries is in essence qualitative and often subjective, and thus cannot be determined by computers alone. FAST hence leverages the verification and responses from users to help determine the query accuracy. In the performance evaluation, FAST provides the query results to the relevant 1,000 users who will give their feedbacks.

B. Results and Analysis

1) Index Construction: Figure 3 shows the index construction latency that consists of two parts, i.e., feature representation and index storage. The feature representation part in turn includes the time spent on detection, representation and matching of interest points, as described in Section III-B. The index storage part is composed of the time spent by the identification and the storage operations of correlated images.

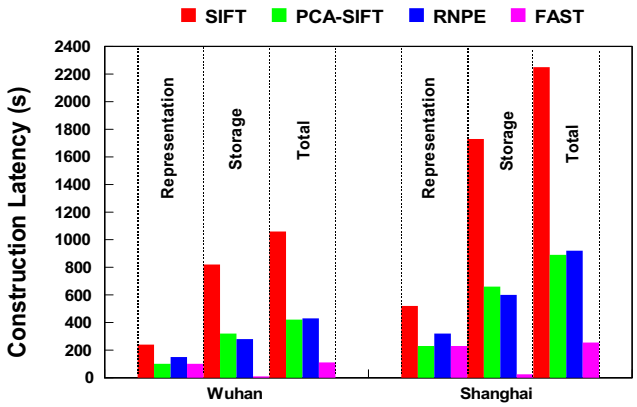


Fig. 3. Index construction latency.

For the feature representation latency, we observe that SIFT performs the worst, i.e., with 240.2s and 520.6s respectively in the Wuhan and Shanghai image datasets, due to its exhaustive feature extraction and point-by-point comparisons. Quite different from it, both PCA-SIFT and FAST leverage light-weight principal components analysis to triage and filter out outliers and loosely correlated points. This results in a significantly reduced number of points required for representation and matching computation, leading to the shortest latency, i.e., 101.8s and 230.5s respectively in the Wuhan and Shanghai image datasets. On the other hand, since RNPE needs to retrieve geographic tags and identify the proximity in an R-tree with $O(\log n)$ complexity, it requires more processing time than PCA-SIFT and FAST, i.e., 152.7s and 328.6s respectively in the two image datasets.

For the index storage latency, SIFT also performs the worst, with up to 825.3s and 1782.6s respectively in the Wuhan and Shanghai image datasets, because it must conduct brute-force-like feature comparisons to identify correlated images. In the meantime, it uses an SQL-based database to store the physical addresses and other metadata information of correlated images, which causes frequent I/O accesses to the low-speed disks.

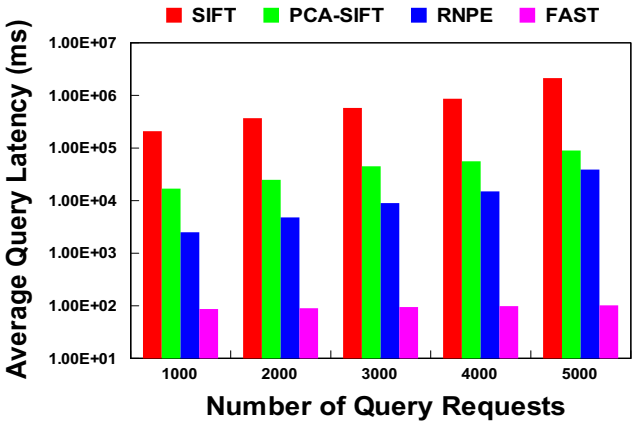
Dealing with a much smaller number of interest points, PCA-SIFT decreases the latency to 327.9s and 661.8s respectively in the Wuhan and Shanghai image datasets. RNPE requires 284.3s and 601.9s respectively in the Wuhan and Shanghai image datasets to complete the storage operations, by storing data in the multi-dimensional R-tree structure.

Overall, FAST performs far better than any of the other schemes. Combining feature representation and index storage latencies, FAST outperforms PCA-SIFT by 75.8% (Wuhan) and 71.3% (Shanghai), and RNPE by 74.2% (Wuhan) and 72.3% (Shanghai). The main reason behind FAST's superior performance lies in its use of space-efficient Bloom filters to represent the feature summarization and correlation-aware LSH computation to improve the performance of correlation identification and index storage.

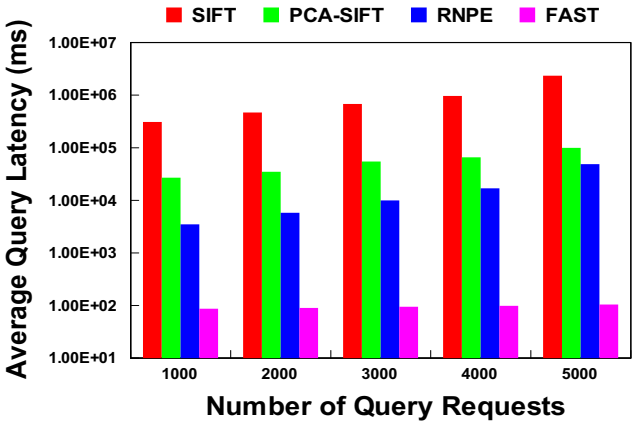
2) Query Latency: Figure 4 shows the average query latency. The query latency includes the computation time of descriptors, e.g., image gradients and SIFT, as described in Section III-B. We examine query performance as a function of the number of simultaneous requests from 1000 to 5000 with an increment of 1000. The latency of PCA-SIFT, at 2min, is one order of magnitude better than SIFT's 35.8min, due to its PCA property. However, SIFT and PCA-SIFT rely on brute-force-like matching to identify similar features that are then stored into an SQL-based database. Their space inefficiency causes frequent disk I/Os, leading to long query latency. We also observe that RNPE performs better when the number of query requests is smaller (e.g., smaller than 1000) but its performance degrades noticeably, as the number of query requests increases, to as long as 55s. This is because the high-complexity MNPG identification algorithm and the R-tree based $O(\log n)$ query complexity of RNPE [34]. The query latency of FAST is much shorter than any of the other schemes and remains roughly at 102.6ms for all datasets and numbers of queries, making *FAST more than 3 orders of magnitude faster than PCA-SIFT and 2 orders of magnitude faster than RNPE.*

The reasons for FAST's advantage are threefold. First, FAST leverages principal components analysis for dimensionality reduction and obtains compact feature vectors. The number of dimensions to be processed is considerably reduced, which in turn lowers the space overhead. Second, the Bloom filter-based summarization further simplifies the representation of feature vectors, which allows us to put more vectors into the main memory. Third, FAST uses cuckoo hashing flat-structured addressing to obtain $O(1)$ real-time query performance.

3) Query Accuracy: Table III shows the query accuracy of all evaluated schemes normalized to that of SIFT, which is one of the state-of-the-art exact-matching approaches and thus serves as the baseline, i.e., 100% accuracy in this metric. Since RNPE leverages simple but error-prone tags to identify similar images, it has the lowest accuracy. PCA-SIFT, on the other hand, uses compact feature vectors and carries out dimensionality reduction, which helps it reduce the number of dimensions to be processed but at a negligible cost of accuracy and results in an accuracy of 99.9983% on average. The accuracy of FAST is around 99.995%, slightly lower than PCA-SIFT. The reason is the possible hash collisions in Bloom filters and LSH. The accuracy of FAST is around 0.005% lower than PCA-SIFT in the Wuhan image dataset. Considering FAST's significant superiority in the search-latency performance (by up to 3 orders of magnitude), we argue that such insignificant loss in accuracy is acceptable, especially for near real-time applications.



(a) Wuhan Dataset.



(b) Shanghai Dataset.

Fig. 4. The average query latency.

TABLE III. QUERY ACCURACY NORMALIZED TO SIFT.

Dataset	Number of Queries	SIFT	PCA-SIFT	RNPE	FAST
Wuhan	1000	100%	99.9995%	97.3%	99.999%
	2000	100%	99.9992%	96.5%	99.997%
	3000	100%	99.9984%	95.9%	99.995%
	4000	100%	99.9977%	94.1%	99.994%
	5000	100%	99.9965%	93.5%	99.990%
Shanghai	1000	100%	99.9992%	96.3%	99.998%
	2000	100%	99.9988%	95.3%	99.994%
	3000	100%	99.9982%	94.2%	99.991%
	4000	100%	99.9969%	93.5%	99.988%
	5000	100%	99.9957%	92.5%	99.986%

4) Space Overhead: Table IV summarizes space overheads of SIFT, PCA-SIFT, RNPE and FAST, normalized to that of SIFT. By reducing the number of dimensions to be processed, PCA-SIFT consumes about 80% of SIFT's space overhead. Since RNPE uses tags, rather than features, to label, represent and identify similar images, it consumes only 50% of storage space of SIFT. Finally, since FAST only needs to maintain the

feature summarization, its space efficiency allows it to consume only about 10% of the space overhead required by SIFT. Thus, FAST is able to store more index information into the main memory and significantly improve query performance.

TABLE IV. SPACE OVERHEAD NORMALIZED TO THAT OF SIFT.

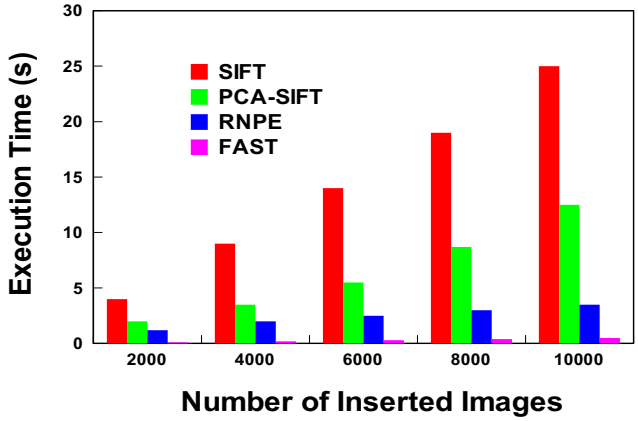
Image Datasets	SIFT	PCA-SIFT	RNPE	FAST
Wuhan	1	0.82	0.58	0.14
Shanghai	1	0.73	0.45	0.11

Our proposed scheme is space-efficient via summarization and compression as described in Section III-A. Hence, we can place the entire index structure of images into the main memory. This space-efficient structure meets the needs of in-memory computing and achieves better query performance than conventional in-memory caching that still needs to access the low-speed disks with some probability. Although a flash disk can alleviate the performance gap, the capacities of flash disks are insufficient to contain the index structures, if FAST is not used, thus still resulting in slow accesses to disks.

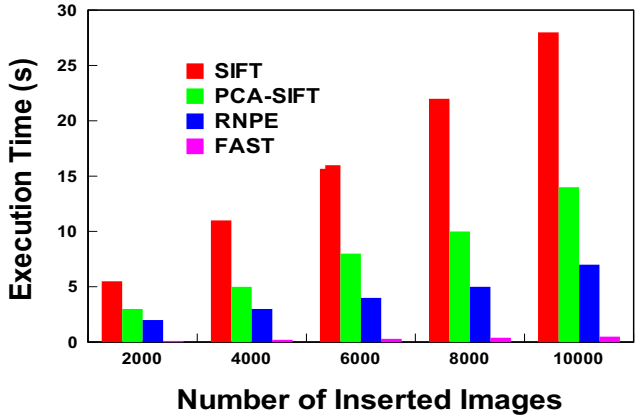
5) Insertion Latency: We examine the latency of inserting new images in Figure 5. For the Wuhan dataset, inserting 10,000 new images consumes 25.8s in SIFT, 12.7s in PCA-SIFT, 3.5s in RNPE and 0.5s in FAST. FAST efficiently locates the new image by exploiting its feature summarization and executing rapid semantic aggregation. Moreover, as the number of inserted images increases, while the insertion latency for SIFT and PCA-SIFT increases almost linearly, FAST's insertion latency remains relatively flat. This is because, although FAST incurs feature-extraction costs that are similar to SIFT and PCA-SIFT, it experiences very little extra latency in storing and indexing these features by virtue of its $O(1)$ complexity LSH. This is a salient feature of FAST in managing the current big data store. For the Shanghai image dataset, we obtain the similar conclusions based on similar experimental results.

6) Rehash Probability: Since hash collisions are unavoidable for any hash functions, rehashing is thus possible in FAST when hash collisions occur. More specifically, rehashing is required in FAST when an endless loop forms in the recursive cuckoo hashing process during the item-insertion operation, which in turn renders the insertion operation a failure. In other words, rehash probability is equal to the failure probability of the insertion operation in FAST. Owing to the flat-structured cuckoo hashing scheme employed, however, FAST is able to significantly reduce the rehashing probability from that of the standard cuckoo hashing. To evaluate FAST for its rehash probability and compare it with the standard cuckoo hashing, we present the experimental results by plotting the insertion-failure probability as a function of the number of items inserted in Figure 6. The average failure probability of FAST is 3 orders of magnitude smaller than that of the standard cuckoo hashing, 1.61×10^{-6} for FAST vs. 3.6×10^{-3} for the standard cuckoo hashing in the Wuhan dataset, and 1.77×10^{-6} for FAST and 4.8×10^{-3} for the standard cuckoo hashing in the Shanghai dataset. In other words, on average, one insertion failure will occur in FAST for several millions of successful insertions, in contrast to one such a failure in only thousands of successful insertions with the standard cuckoo hashing.

7) Multicore-enabled Parallel Queries: The FAST design has the salient feature of supporting parallel query operations



(a) Wuhan Dataset.



(b) Shanghai Dataset.

Fig. 5. The latency of inserting images.

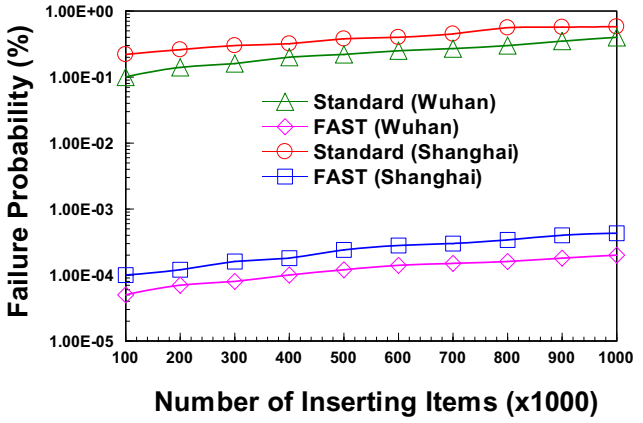


Fig. 6. Insertion failure (rehash) probability.

via its flat-structured addressing of cuckoo hashing that helps expose sufficient amount of query parallelism. Figure 7 shows the latency of queries carried out on a multicore-CPU based system as a function of the number of cores. We observe that the query latency decreases almost linearly with the increase in the number of cores. This linear speedup is mainly attributed to FAST's property of flat-structured addressing. We believe

that the query performance can be further improved if more cores are used.

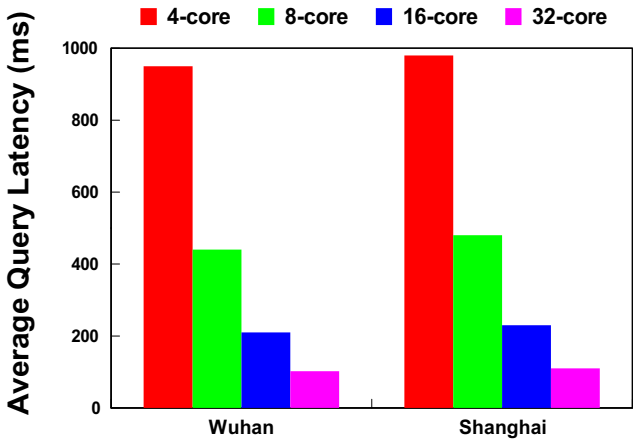


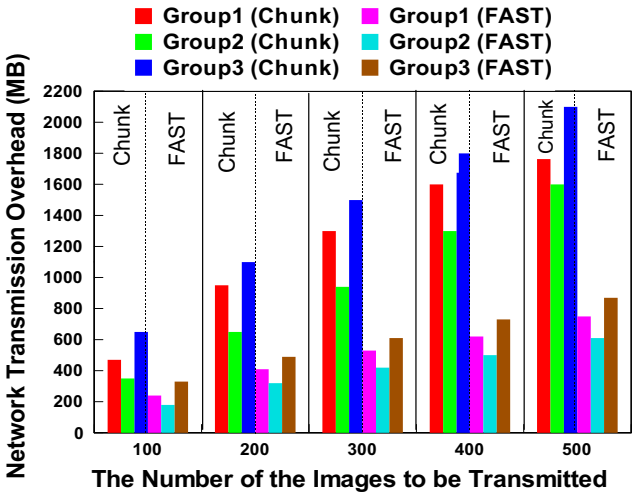
Fig. 7. Multi-core based query latency.

8) User Experiences from Smartphones: In FAST's Android-based clients, we designed a friendly and easy-to-use interface for users to upload images and submit queries. To support local image processing, we ported an open-source implementation of PCA-SIFT feature extraction algorithm to Android. Moreover, in order to comprehensively evaluate the performance, we divide 1,000 users who use this client in their smartphones into 3 groups based on their crowdsourcing interests (i.e., approximately equal number of the landmarks of disaster zones in the image sets). Users download and install FAST's client application software that offers the functions of image identification and energy-efficient network transmission as shown in Figure 8. We compare FAST with the Chunk-based scheme due to its energy efficiency, which has been examined and recommended by the evaluation of battery power consumption with 11 Internet applications [35].

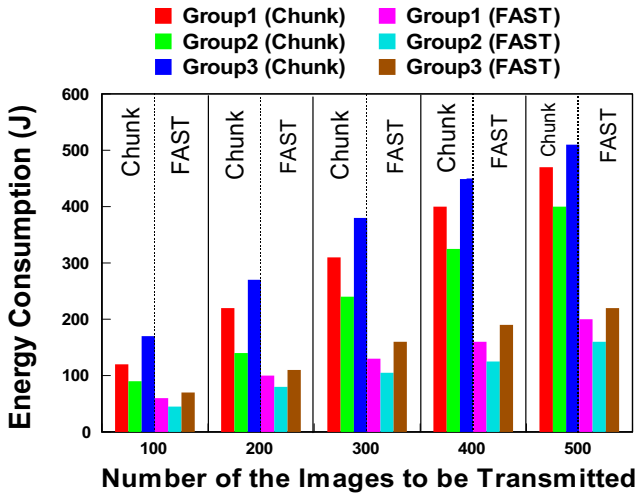
In a common case, a smartphone needs to upload all images to the destination server via wireless data transmission and requires continuous bandwidth guarantee, a stringent requirement that is difficult to meet in a crowdsourcing environment. FAST leverages its near-duplicate identification technique to significantly reduce the amount of images to be transmitted. Figure 8(a) shows the network transmission overhead by examining the practical use of bandwidth in transmitting a batch of images.

We have two observations from the results. First, compared with chunk-based transmission scheme, FAST can achieve more than 55.2% bandwidth savings due to the significantly decreased amount of images to be transmitted. Second, we observe that the percentage of bandwidth savings will increase with the increasing number of images. This is because with more images there is a higher probability of images being similar. These results also demonstrate the scalability of FAST.

To measure energy consumption, we use the Monsoon Power Monitor [36] and run the experiments of uploading and sharing the interested images. The Monsoon Power Monitor is configured by blocking the positive terminal on the phone's battery with electrical tape. The voltage normally supplied by the battery is supplied by the monitor. It records voltage and



(a) Network transmission overhead.



(b) Energy consumption.

Fig. 8. User Experiences from Smartphones.

current with a sample rate of 6 kHz. During our experiments, the screen is set to stay in the awake mode with constant brightness and auto-rotate screen off. All radio communication is disabled except for WiFi.

Figure 8(b) shows the energy consumption with the increase in the number of the transmitted images. We observe that, compared with the chunk-based transmission scheme, the FAST scheme can achieve from 46.9% to 62.2% energy savings in the three user groups due to the significantly decreased numbers of the images to be transmitted. Moreover, the percentage of energy savings is consistent with that of bandwidth savings since fewer transmitted images consume less energy. These results show that FAST offers an energy-saving benefit to some smartphone applications.

V. Related Work

In this section, we present a brief survey of recent studies in the literature most relevant to the FAST research from

the aspects of data analytics, searchable file systems and deduplication-based redundancy detection.

Data Analytics. Data analytics has received increasing attention from both industrial and academic communities. In order to bridge the semantic gap between the low-level data contents and the high-level user understanding of the system, a behavior-based semantic analysis framework [37] is proposed, which includes an analysis engine for extracting instances of user-specified behavior models. ISABELA-QA [38] is a parallel query processing engine that is designed and optimized for analyzing and processing spatiotemporal, multivariate scientific data. MixApart [39] uses an integrated data caching and scheduling solution to allow MapReduce computations to analyze data stored on enterprise storage systems. The frontend caching layer enables the local storage performance required by data analytics. The shared storage back-end simplifies data management. Three common analysis techniques [40], including topological analysis, descriptive statistics, and visualization, are explored to support efficient data movement between in-situ and in-transit computations. In this context, FAST is a useful tool that complements and improves the existing schemes to obtain correlated affinity from near duplicate images and execute semantic grouping to support fast query service.

Searchable File Systems. Spyglass [22] exploits the locality of file namespace and skewed distribution of metadata to map the namespace hierarchy into a multi-dimensional K-D tree and uses multilevel versioning and partitioning to maintain consistency.Glance [41], a just-in-time sampling-based system, can provide accurate answers for aggregate and top-k queries without prior knowledge. SmartStore [24] uses Latent Semantic Indexing (LSI) tool [42], [43] to aggregate semantically correlated files into groups and support complex queries. Ceph [44] and its demonstration system [45] use dynamic subtree partition to avoid metadata-access hot spots and support filename-based query. FastQuery [46] is a software framework that utilizes a FastBit based index and query technology to process massive datasets on modern supercomputing platforms. Locality-Sensitive Bloom Filter [47] proposes a locality-aware and space-efficient data structure that can efficiently support the in-memory computing. SciHadoop [48] executes queries as map/reduce programs defined over the logical data model to reduce total data transfers, remote reads, and unnecessary reads. Unlike these approaches, FAST offers the salient features of querying near duplicate images in a near real-time manner.

Deduplication based Redundancy Detection. DDFS [49] proposes the idea of exploiting the backup-stream locality to reduce network bandwidth and accesses to on-disk index. Extreme Binning [50] exploits the file similarity for deduplication and can be applied to non-traditional backup workloads with low-locality (e.g., incremental backup). ChunkStash [29] maintains the chunk fingerprints in an SSD instead of a hard disk to accelerate the lookups. SiLo [51] is a near-exact deduplication system that exploits both similarity and locality to achieve high duplicate elimination and throughput with low RAM overheads. The cluster-based deduplication [52] examines the tradeoffs between stateless data routing approaches with low overhead and stateful approaches with high overhead but being able to avoid imbalances. Sparse Indexing [53] exploits the inherent backup-stream locality to solve the index-lookup bottleneck problem. Moreover, by exploiting similarities between files or versions of the same file, LBFS [54] is shown to be a low-bandwidth network file system. The potential of data deduplication in HPC centers is presented in [55] via quantitative analysis on the potential for capacity reduction for 4 data centers. In order to opportunistically leverage resources on end hosts, EndRE [56] uses a fingerprinting scheme called SampleByte that is much faster than Rabin fingerprinting while delivering similar compression gains. In contrast to these existing system-level approaches, FAST provides both application-level and system-level detection for both identical and near duplicate data. FAST can meet the needs of handling the rapid growth of big data in an efficient manner.

VI. Conclusion

This paper proposes a near real-time scheme, called FAST, to support efficient and cost-effective searchable data analytics in the cloud. FAST is designed to exploit the correlation property of data by using correlation-aware hashing and manageable flat-structured addressing. This enables FAST to significantly reduce processing latency of correlated file detection with acceptably small loss of accuracy. We discuss how the FAST methodology can be related to and used to enhance some storage systems, including Spyglass and SmartStore, as well as a use case. FAST is demonstrated to be a useful tool in supporting near real-time processing of real-world data analytics applications.

Acknowledgements

This work was supported in part by National Natural Science Foundation of China (NSFC) under Grant 61173043, National Basic Research 973 Program of China under Grant 2011CB302301, NSFC under Grant 61025008, and US NSF under Grants NSF-CNS-1016609 and NSF-CNS-1116606. The authors are grateful to anonymous reviewers and our shepherd, Mahmut Kandemir, for their feedback and guidance.

References

[1] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "A view of cloud computing," *Communications of the ACM*, vol. 53, no. 4, pp. 50–58, 2010.

[2] A. Marathe, R. Harris, D. K. Lowenthal, B. R. de Supinski, B. Rountree, M. Schulz, and X. Yuan, "A comparative study of high-performance computing on the cloud," *Proc. HPDC*, 2013.

[3] P. Nath, B. Urgaonkar, and A. Sivasubramaniam, "Evaluating the usefulness of content addressable storage for high-performance data intensive applications," *Proc. HPDC*, 2008.

[4] Gartner, Inc., "Forecast: Consumer digital storage needs, 2010-2016," 2012.

[5] Storage Newsletter, "7% of consumer content in cloud storage in 2011, 36% in 2016," 2012.

[6] J. Gantz and D. Reinsel, "The Digital Universe in 2020: Big Data, Bigger Digital Shadows, and Biggest Growth in the Far East," *International Data Corporation (IDC) iView*, December 2012.

[7] Y. Ke and R. Sukthankar, "PCA-SIFT: A more distinctive representation for local image descriptors," *Proc. CVPR*, 2004.

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

[9] J. Liu, Z. Huang, H. T. Shen, H. Cheng, and Y. Chen, "Presenting Diverse Location Views with Real-time Near-duplicate Photo Elimination," *Proc. ICDE*, 2013.

[10] Changewave research *http://www.changewaveresearch.com, 2011*.

[11] P. Indyk and R. Motwani, "Approximate nearest neighbors: towards removing the curse of dimensionality," *Proc. STOC*, pp. 604–613, 1998.

[12] R. Pagh and F. Rodler, "Cuckoo hashing," *Proc. ESA*, pp. 121–133, 2001.

[13] M. Samadi, A. Jamshidi, J. Lee, and S. Mahlke, "Paraprox: Pattern-Based Approximation for Data Parallel Applications," *Proc. ASPLOS*, 2014.

[14] S. Venkataramani, V. K. Chippa, S. T. Chakradhar, K. Roy, and A. Raghunathan, "Quality programmable vector processors for approximate computing," in *Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture*, ACM, 2013.

[15] M. Samadi, J. Lee, D. A. Jamshidi, A. Hormati, and S. Mahlke, "Sage: self-tuning approximation for graphics engines," in *Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 13–24, ACM, 2013.

[16] A. Sampson, J. Nelson, K. Strauss, and L. Ceze, "Approximate storage in solid-state memories," in *Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 25–36, ACM, 2013.

[17] A. M. Khan, D. F. Gleich, A. Pothen, and M. Halappanavar, "A multithreaded algorithm for network alignment via approximate matching," *Proc. International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, 2012.

[18] J. Kim, D. Chae, J. Kim, and J. Kim, "Guide-copy: fast and silent migration of virtual machine for datacenters," *Proc. International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, 2013.

[19] M. Lassnig, T. Fahringer, V. Garonne, A. Molfetas, and M. Barisits, "A similarity measure for time, frequency, and dependencies in large-scale workloads," *Proc. International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, 2011.

[20] S. Doraimani and A. Iamnitchi, "File grouping for scientific data management: lessons from experimenting with real traces," *Proc. HPDC*, 2008.

[21] Y. Hua, H. Jiang, Y. Zhu, D. Feng, and L. Tian, "Semantic-Aware Metadata Organization Paradigm in Next-Generation File Systems," *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, vol. 23, no. 2, pp. 337–344, 2012.

[22] A. W. Leung, M. Shao, T. Bisson, S. Pasupathy, and E. L. Miller, "Spyglass: Fast, Scalable Metadata Search for Large-Scale Storage Systems," *Proc. FAST*, 2009.

[23] Y. Hua, H. Jiang, Y. Zhu, D. Feng, and L. Xu, "SANE: Semantic-Aware Namespace in Ultra-large-scale File Systems," *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, vol. 25, no. 5, pp. 1328–1338, 2014.

[24] Y. Hua, H. Jiang, Y. Zhu, D. Feng, and L. Tian, "SmartStore: A New Metadata Organization Paradigm with Semantic-Awareness for Next-Generation File Systems," *Proc. SC*, 2009.

[25] D. Lowe, "Distinctive image features from scale-invariant keypoints," *International Journal of Computer Vision*, vol. 60, no. 2, pp. 91–110, 2004.

[26] B. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Communications of the ACM*, vol. 13, no. 7, pp. 422–426, 1970.

[27] A. Andoni and P. Indyk, "Near-Optimal Hashing Algorithms for Approximate Nearest Neighbor in High Dimensions," *Communications of the ACM*, vol. 51, no. 1, pp. 117–122, 2008.

[28] Q. Lv, W. Josephson, Z. Wang, M. Charikar, and K. Li, "Multi-probe LSH: Efficient indexing for high-dimensional similarity search," *Proc. VLDB*, pp. 950–961, 2007.

[29] B. Debnath, S. Sengupta, and J. Li, "ChunkStash: speeding up inline storage deduplication using flash memory," *Proc. USENIX ATC*, 2010.

[30] D. Lowe, "Object recognition from local scale-invariant features," *Proc. IEEE ICCV*, 1999.

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

[32] M. Datar, N. Immorlica, P. Indyk, and V. Mirrokni, "Locality-sensitive hashing scheme based on p-stable distributions," *Proc. Annual Symposium on Computational Geometry*, 2004.

[33] Y. Tao, K. Yi, C. Sheng, and P. Kalnis, "Quality and Efficiency in High-dimensional Nearest Neighbor Search," *Proc. SIGMOD*, 2009.

[34] A. Guttman, "R-trees: A dynamic index structure for spatial searching," *Proc. ACM SIGMOD*, pp. 47–57, 1984.

[35] Y. Liu, L. Guo, F. Li, and S. Chen, "An empirical evaluation of battery power consumption for streaming data transmission to mobile devices," in *Proc. Multimedia*, pp. 473–482, 2011.

[36] Monsoon power monitor *http://www.msoon.com, 2012*.

[37] A. Viswanathan, A. Hussain, J. Mirkovic, S. Schwab, and J. Wroclawski, "A semantic framework for data analysis in networked systems," *Proc. NSDI*, 2011.

[38] S. Lakshminarasimhan, J. Jenkins, I. Arkatkar, Z. Gong, H. Kolla, S.-H. Ku, S. Ethier, J. Chen, C.-S. Chang, S. Klasky, *et al.*, "ISABELA-QA: query-driven analytics with ISABELA-compressed extreme-scale scientific data," *Proc. SC*, 2011.

[39] M. Mihailescu, G. Soundararajan, and C. Amza, "MixApart: decoupled analytics for shared storage systems," *Proc. FAST*, 2013.

[40] J. C. Bennett, H. Abbasi, P.-T. Bremer, R. Grout, A. Gyulassy, T. Jin, S. Klasky, H. Kolla, M. Parashar, V. Pascucci, *et al.*, "Combining in-situ and in-transit processing to enable extreme-scale scientific analysis," *Proc. SC*, 2012.

[41] H. Huang, N. Zhang, W. Wang, G. Das, and A. Szalay, "Just-In-Time Analytics on Large File Systems," *Proc. FAST*, 2011.

[42] S. Deerwester, S. Dumas, G. Furnas, T. Landauer, and R. Harsman, "Indexing by latent semantic analysis," *J. American Society for Information Science*, pp. 391–407, 1990.

[43] C. Papadimitriou, P. Raghavan, H. Tamaki, and S. Vempala, "Latent Semantic Indexing: A Probabilistic Analysis," *Journal of Computer and System Sciences*, vol. 61, no. 2, pp. 217–235, 2000.

[44] S. Weil, S. A. Brandt, E. L. Miller, D. D. E. Long, and C. Maltzahn, "Ceph: A scalable, high-performance distributed file system," *Proc. OSDI*, 2006.

[45] C. Maltzahn, E. Molina-Estolano, A. Khurana, A. J. Nelson, S. A. Brandt, and S. Weil, "Ceph as a Scalable Alternative to the Hadoop Distributed File System," *;login: The USENIX Magazine*, August 2010.

[46] J. Chou, K. Wu, O. Rubel, M. Howison, J. Qiang, B. Austin, E. W. Bethel, R. D. Ryne, A. Shoshani, *et al.*, "Parallel index and query for large scale data analysis," *Proc. SC*, 2011.

[47] Y. Hua, B. Xiao, B. Veeravalli, and D. Feng, "Locality-Sensitive Bloom Filter for Approximate Membership Query," *IEEE Transactions on Computers*, vol. 61, no. 6, pp. 817–830, 2012.

[48] J. B. Buck, N. Watkins, J. LeFevre, K. Ioannidou, C. Maltzahn, N. Polyzotis, and S. Brandt, "SciHadoop: Array-based query processing in hadoop," *Proc. SC*, 2011.

[49] B. Zhu, K. Li, and H. Patterson, "Avoiding the disk bottleneck in the data domain deduplication file system," *Proc. FAST*, 2008.

[50] D. Bhagwat, K. Eshghi, D. Long, and M. Lillibridge, "Extreme binning: Scalable, parallel deduplication for chunk-based file backup," *Proc. IEEE MASCOTS*, 2009.

[51] W. Xia, H. Jiang, D. Feng, and Y. Hua, "SiLo: A Similarity-Locality based Near-Exact Deduplication Scheme with Low RAM Overhead and High Throughput," *Proc. USENIX ATC*, 2011.

[52] W. Dong, F. Douglis, K. Li, H. Patterson, S. Reddy, and P. Shilane, "Tradeoffs in scalable data routing for deduplication clusters," *Proc. FAST*, 2011.

[53] M. Lillibridge, K. Eshghi, D. Bhagwat, V. Deolalikar, G. Trezise, and P. Camble, "Sparse indexing: large scale, inline deduplication using sampling and locality," *Proc. FAST*, 2009.

[54] A. Muthitacharoen, B. Chen, and D. Mazieres, "A low-bandwidth network file system," *Proc. SOSP*, 2001.

[55] D. Meister, J. Kaiser, A. Brinkmann, T. Cortes, M. Kuhn, and J. Kunkel, "A study on data deduplication in hpc storage systems," *Proc. SC*, 2012.

[56] B. Aggarwal, A. Akella, A. Anand, A. Balachandran, P. Chitnis, C. Muthukrishnan, R. Ramjee, and G. Varghese, "EndRE: an end-system redundancy elimination service for enterprises," *Proc. NSDI*, 2010.