Real-Time Semantic Search Using Approximate Methodology for Large-Scale Storage Systems

Yu Hua, *Senior Member, IEEE*, Hong Jiang, *Fellow, IEEE*, and Dan Feng, *Member, IEEE*

Abstract—The challenges of handling the explosive growth in data volume and complexity cause the increasing needs for semantic queries. The semantic queries can be interpreted as the correlation-aware retrieval, while containing approximate results. Existing cloud storage systems mainly fail to offer an adequate capability for the semantic queries. Since the true value or worth of data heavily depends on how efficiently semantic search 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 semantic queries based 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. FAST is further improved by using semantic-aware namespace to provide dynamic and adaptive namespace management for ultra-large storage systems. Extensive experimental results demonstrate the efficiency and efficacy of FAST in the performance improvements.

Index Terms—Cloud storage, data analytics, real-time performance, semantic correlation

1 Introduction

Cloud storage systems generally contain large amounts of data that critically require fast and accurate data retrieval to support intelligent and adaptive cloud services [1], [2], [3]. For example, 7 percent of consumers stored their contents in the cloud in 2011, and the figure will grow to 36 percent 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 (DA) tools. IDC estimates that by 2020, as much as 33 percent of all data will contain information that might be valuable if analyzed [6]. Hence, efficient data analytics are important.

Existing content-based analysis tools not only cause high complexity and costs, but also fail to effectively handle the massive amounts of 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 [7].

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:

High access latency. Existing approaches to unstructured data search and 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 [8], results in 200 KB worth of features on average. This means that analyzing 1 million such images will lead to approximately 200 GB of storage space requirement just for

- *Y. Hua and D. Feng are with the Wuhan National Laboratory for Optoelectronics, School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan, China. E-mail: {csyhua, dfeng}@hust.edu.cn.*
- *H. Jiang is with the Department of Computer Science and Engineering, University of Nebraska-Lincoln, Lincoln, NE 68588. E-mail: jiang@cse.unl.edu.*

Manuscript received 27 Dec. 2014; revised 25 Mar. 2015; accepted 13 Apr. 2015. Date of publication 21 Apr. 2015; date of current version 16 Mar. 2016. Recommended for acceptance by Z. Lan.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TPDS.2015.2425399

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 [9], [10].

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) [11]. 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 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.

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.

In order to support efficient data analytics in the cloud, real-time processing approaches 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 [12] and flat-structured addressing [13] 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 and high-end computing. 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.

Space-efficient summarization. FAST leverages a Bloom-filter based summarization representation that has the salient features of simplicity and ease of use. We hash the large-size vectors of files into space-efficient Bloom filters to efficiently and effectively identify similar files in a real-time manner. 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.

Energy efficiency via hashing. In order to substantially reduce the amount of similar images to be transmitted, FAST improves the energy efficiency in the smartphones via a near-deduplication scheme. Our design alleviates the computation overheads of existing schemes for similarity detection of files by using locality-sensitive hashing (LSH) [12] 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.

Semantic-aware namespace. By exploiting semantic correlations among files, FAST leveraging SANE [14] to dynamically aggregate correlated files into small, flat but readily manageable groups to achieve fast and accurate lookups. Moreover, in the context of semantic-aware namespace, 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. Moreover,

Real system implementation. In order to comprehensively evaluate the system performance, 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 200 TB 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 2.1 presents the results of a user survey, as well as the FAST methodology. Section 3 describes the FAST architecture and implementation details. We evaluate the FAST performance in Section 4. Section 5 presents the related work. Section 6 concludes the paper.

2 FAST METHODOLOGY

In order to improve query efficiency and reduce operation cost in smartphones, we need to reduce the redundant data,

TABLE 1
Survey Results

		Answer	PCT	Answer	PCT	Answer	PCT	Answer	PCT	Answer	PCT
Data Attributes	Total size	(0, 1 MB)	0.1%	(1 MB, 100 MB)	11.5%	(100 MB, 10 GB)	22.5%	(10 GB, 1 TB)	47.5%	(1 TB, 100 TB)	18.4%
	Average file size	(0, 10 MB)	55.2%	(10 MB, 100 MB)	24.8%	(100 MB, 1 GB)	10.2%	(1 GB, 10 GB)	7.6%	(10 GB, 100 GB)	2.2%
	Number of formats	(1, 10)	7.5%	(10, 50)	46.8%	(50, 100)	37.9%	(100, 500)	4.2%	more	3.6%
Task Attributes	Execution time	(1 s, 1 min)	5.5%	(1 min, 1 hour)	22.5%	(1 hour, 1 day)	40.5%	(1 day, 1 week)	21.5%	more than 1 week	10%
	Resource bottleneck	CPU	17.5%	memory	42.8%	hard disks	2.5%	SSD	14.4%	network	22.8%
General Concerns	Metric of importance	Accuracy	28.2%	Time effi.	39.2%	Space effi.	3.5%	Energy effi.	10.6%	Costs	18.5%
	Acceptable accuracy (%)	(0, 80)	4.2%	(80, 90)	12.5%	(90, 95)	28.5%	(95, 100)	45.2	100	9.6%

such as identifying and filtering redundant data at the client side. The data reduction allows users to upload more valuable data in a limited time frame and battery budget, thus increasing the chance of data sharing. 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 [15] concerning smartphone dislikes, 38 percent 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.

2.1 Observations, Insights and Motivations

In this section, we first present a comprehensive survey of administrators/scientists at a cloud center and then study three real-world cases, with a goal of gaining useful insights to motivate the FAST research.

2.1.1 Insights from a Comprehensive Survey of Cloud Users

In order to better understand the requirements from high-performance cloud users, we conducted a user survey to obtain insights and observations that are very helpful to our design and, hopefully, to the cloud storage research community.

The survey was conducted among researchers in the computing center. A total of 200 researchers, including 40 administrators, 90 engineers and 70 scientists, were polled on their opinions on large-scale data analytics (e.g., size, types, computation complexity, average running time, resource consumption, top concerns, acceptable accuracy of analytics results) based on their experiences in using the cloud services. The application domains include computational biology and bioinformatics, computational earth and atmospheric sciences, computational chemistry and computational physics, data analysis and data mining, computational fluid dynamics, computational solid mechanics, medicine and biotechnology computing, materials science, and engineering simulation, etc.

The survey results are summarized in Table 1. Specifically, three categories of questions were asked, i.e., *data attributes* (e.g., average file size, total size and the number of formats), *task attributes* (e.g., execution time and resource bottleneck) and *general concerns* (i.e., metric of importance and acceptable accuracy of results). Answers to the questions are grouped into ranges (i.e., minimum and maximum). PCT in the table indicates the percentage of all users who provide the corresponding answers (in the column on the immediate left). We assume that each person has one dataset.

The survey results help us gain some valuable insights and observations that are summarized below.

Temporal and spatial overheads of large-size indexing are not cost-efficient. The total data size per high-performance cloud application was much larger than 10 GB (around 66 percent) and more than 18 percent of datasets were larger than 1 TB, in 2012. This observation is consistent with the *Science* poll [16] in 2011, in which 7 percent of datasets exceeded 1 TB. While taking into account the rapid growth of datasets, this percentage increase is reasonable. Moreover, we observe that a large fraction (55.2 percent) of the data in the datasets is stored in the form of small files (smaller than 10 MB per file). This trend implies that a disproportionally large size of index structure must be dedicated to small files and consumes substantial space in the main memory.

Real-time cloud applications require fast responses. Over 71 percent of tasks require more than 1 hour execution time, of which 10 percent tasks run for one week or more. This observation demonstrates the importance of real-time processing. We discuss with the researchers about the reasons. The main reasons, according to the researchers, are twofold.

The first is that, consistent with our first insight above, index structures of these cloud applications are too big to fit entirely in the high-speed main memory, even if the allocated main memory size is up to the TB scale. The ensuing frequent disk I/Os and network transmissions further aggravate the execution performance. Second, some applications encounter occasional system crashes, which leads to re-computation that substantially lengthens the latency. Clearly, *since the most expensive resource is memory and it is the insufficient memory space for index structures that causes most of the execution delays, space-efficient index structures are a key to delivering the high performance required for (near-) real-time data analytics*.

Willingness to trade small loss of accuracy of results for significantly improved performance. Although 28.2 percent respondents select accuracy as the most important concern, around 40 percent choose time efficiency as their topmost concern. We further asked how much accuracy is acceptable if 100 percent accuracy can not be guaranteed. Only 9.6 percent respondents insist on 100 percent accuracy, while 73.7 percent can accept more than 90 percent accuracy. These results suggest that *approximate data analytics are often acceptable and there is a willingness to trade acceptably small loss of accuracy for significantly improved performance*.

2.1.2 One Case

Although surveillance systems are helpful in finding clues, facial recognition is often difficult to use in large-scale investigations because surveillance footage often does not have full-frontal images or sufficient clarity due to the relatively low image resolution, a basic requirement for computers to identify the necessary key points on a face with the state-of-the-art facial recognition techniques [17], [18], [19], [20], [21]. Fortunately, citizens are playing an increasing role in investigations as a result of the growing culture of crowd-sourcing of individually taken high-resolution pictures, with the latest example of this being the investigation. Photographs from average citizens, news organizations and others have played a helpful role in identifying useful clues during the investigation. In fact, combining forensic image data from professional and personal sources has worked previously as well. In 2011, authorities used nearly 1 million digital images and 1,600 hours of video gathered from the public and closed-circuit cameras to identify acts. Now, more than a billion people carry a basic tool of surveillance in their pockets, namely, their camera-equipped smartphones and portable cameras, that can be easily connected to the Internet. Given the increasingly crowd-sourced and growing image data store, it is up to the data analytics and systems support to quickly and accurately identify useful clues, patterns, etc.

Based on and inspired by the survey observations and the real-world case analysis, *our research concentrates on fast and cost-effective approximate data analytics. The main function is to fast identify similar images from the massive image datasets in the cloud*.

2.2 The Methodology

2.2.1 The Idea

The idea behind FAST is to explore and exploit the semantic correlation property within and among datasets via correlation-aware hashing [12] and flat-structured addressing [13] to significantly reduce the processing latency, while incurring acceptably small loss of accuracy, as shown in Fig. 1. Specifically, the correlation-aware hashing is to identify the correlated files via the hash-computing manner, such as locality-sensitive hashing. Moreover, unlike the frequent probing and addressing in the conventional multi-level hierarchy, the flat-structured addressing is to find the queried item by directly probing the bucket.

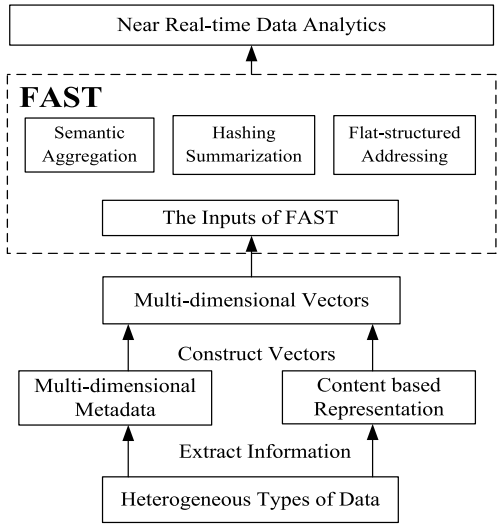


Fig. 1. The FAST methodology for multiple data types.

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. 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 [14] and SmartStore [23], 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.

Most file systems or their traces include the multi-dimensional attributes to support real-time case. For instance, there are five attributes in the HP trace (i.e., file name, device ID, last modified time, user ID, file size) [24], five attributes in the MSN trace (i.e., user ID, device ID, file size, filename, executed time) [25], four attributes in the EECS trace (i.e., user ID, device ID, file size, filename) [26] and four attributes in the Google clusters trace (i.e., access time, job ID, task ID and job Type) [27].

2.2.2 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

TABLE 2
The Relationship and Correspondence between the FAST Methodology and Example System Implementations

	FAST Methodology	Use-case (Images)	Spyglass [22]	SmartStore [23]
Data Analytics	Flat-structured Addressing Semantic Aggregation Hash Summarization	Cuckoo Hashing Storage LSH based Clustering Summary Vectors	Hierarchical Addressing Subtree Partitioning Membership Bloom Filters	Hierarchical Addressing Latent Semantic Indexing Membership Bloom Filters
Vector Extraction	Content Description Metadata Representation	PCA-SIFT Features Vectors	Signature Files K-D Tree	No R-Tree

explained as follows. 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 (SA) 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 2, we elaborate on the corresponding relationship between the modules of the FAST methodology and typical searchable storage systems, such as Spyglass [22] and SmartStore [23], as well as a use case illustrated in Section 3.1. The corresponding relationship includes the vector extraction (VE) for metadata and content, and the data analytics 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.

By leveraging SANE's semantic-aware namespace [14], FAST aims to further improve the scalability, flexibility and entire system performance to achieve the following design goals.

2.2.3 Design Goals of Implementing Semantic-Aware Namespace

The FAST namespace design attempts to achieve the following major goals.

Goal 1: High scalability. By leveraging semantic aggregation, FAST is able to improve entire system scalability. The semantics embedded in file attributes and user access patterns can be used to reveal the potential correlation of file in a large and distributed storage system. These files are thus aggregated into the same or adjacent groups by using the semantic-aware per-file namespace. For future file operations, such as *read*, *write* and *update*, we can carry out these operations within one or a very small number of groups.

Goal 2: Smart namespace. In order to offer smart namespace in FAST, we need to manage the file system namespace in an intelligent and automatic way. In FAST's namespace, we identify semantic correlations and data affinity via lightweight hashing schemes. In a namespace, a file uses its most closely semantically correlated files to build the per-file namespace. One salient feature is that the namespace is flat without hierarchy. In order to accurately represent the namespace, FAST makes use of multi-dimensional, rather than single-dimensional, attributes to identify semantic correlations. FAST hence obtains the accuracy and simplicity in namespace for large-scale file systems.

Goal 3: Efficient compatibility. FAST is designed to be compatible with or orthogonal to existing file systems. We hence implement FAST as a middleware between user applications and file systems. For the file system stacks, FAST is transparent, thus being flexibly used in most file systems to significantly improve system performance.

3 Design and Implementations

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

3.1 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 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.

3.2 The 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. Fig. 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 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 (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 [28] and PCA-SIFT schemes [8] 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.

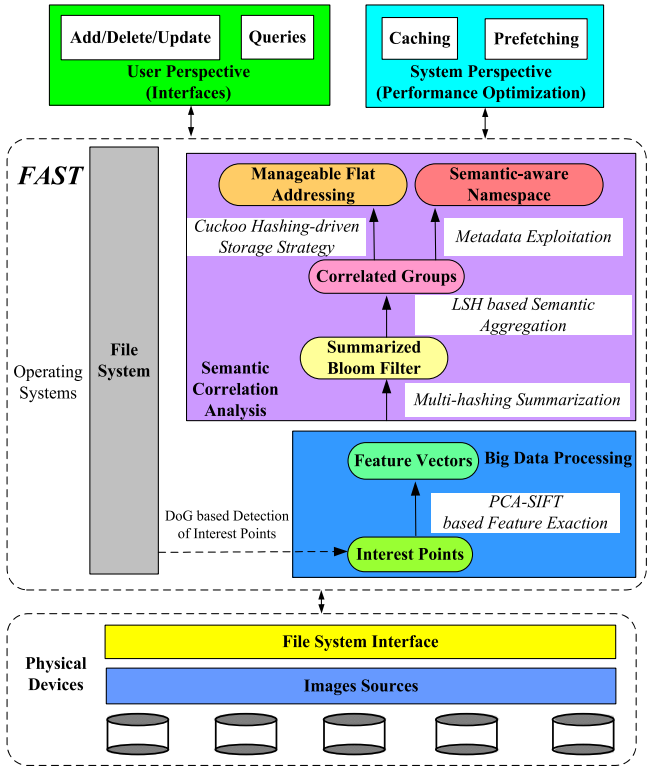


Fig. 2. The FAST implementation of the image-identification use case.

The identified features generally require a relatively large space for representation, for example, 200 KB per 1 MB image in the state-of-the-art PCA-SIFT scheme [8]. One billion such images would thus require about 200 TB 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 [29], 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 [12], [30] 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 3.4.3.

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.

3.3 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 (PCA) 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.

3.4 Semantic-Aware Grouping

3.4.1 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 [29]. 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 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 200 KB to 40 B, a 5000-fold space reduction, with only $O(1)$ computational complexity.

3.4.2 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 [12]. 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 $p, q \in S$*

1) *If $||p, q|| \leq R$ then $Pr_{\mathbb{H}}[h(p) = h(q)] \geq P_1$,*
2) *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 [31].

3.4.3 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 [13]. 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 [13], [32]. 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.

3.5 Semantic-Aware Namespace to Improve System Performance

The namespace serves as a middleware in the file systems by offering an optional semantic-aware function. To be compliant with conventional hierarchical file systems, the user-level client contains two interfaces, which can be decided by the application requirements. If working with the conventional systems, the proposed namespace bypasses the semantic middleware and directly links with the application, like existing file systems. Users can access file systems via the existing POSIX interfaces. Otherwise, the namespace is used via the enhanced POSIX I/O in the user space. By exploiting the semantic correlations existing in the files' metadata, FAST is able to support efficient semantic grouping and allow users to carry out the read/write operations on files via the enhanced POSIX I/O interfaces.

In order to communicate with users or application, FAST leverages the client components, in which the enhanced POSIX system interface is exposed to provide naming and offer complex query services. Moreover, in order to significantly improve the entire system performance, FAST offers both user-level and kernel-level clients. Specifically, the user-level client may choose to directly link with the application or work in the user space FUSE [33]. In the kernel-level client, FAST is mounted into a conventional file system. We allow virtual file system (VFS) operations to be redirected to a user-level daemon. Unlike conventional directory based hierarchy, FAST makes use of the VFS operations to support semantic grouping. We can obtain the data from page cache to further transmit to the daemon. Furthermore, most operations run in the form of the namespace-based tuples. In the tuples, a VFS `readdir` is able to return the names of the encapsulated data. The main operations, such as `create` and `rename`, guarantee that there are no collisions between the claimed namespace and existing tuples. In the kernel module, the operation `lookup` supports the Linux `dentry` working with the inode for a given file name. Other VFS operations, such as `commit_write` and `unlink`, are able to contact with the daemon and update the modified file representation.

4 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

TABLE 3
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

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.

4.1 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 64 GB RAM, a 1,000 GB 7200 RPM hard disk and a Gigabit network interface card. The implementation required approximately 1,200 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.

4.1.1 Evaluation Workload: Real Image Dataset

We collect real and openly accessible 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. The analysis of false positives and negatives can be found in our conference version [34]. 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 200 TB in storage size. The key characteristics of the image dataset are summarized in Table 3. Moreover, the query requests, which are simultaneously issued from 500 clients, consist of the queried portraits in the real datasets.

4.1.2 Evaluation Baselines, Metrics and Parameters

We compare FAST with the state-of-the-art schemes, SIFT [35], PCA-SIFT [8] and real-time near-duplicate photo elimination (RNPE) [10]. 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 for dimensionality reduction to obtain compact feature vectors. We implement scale-invariant feature transform (SIFT) [35], 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.

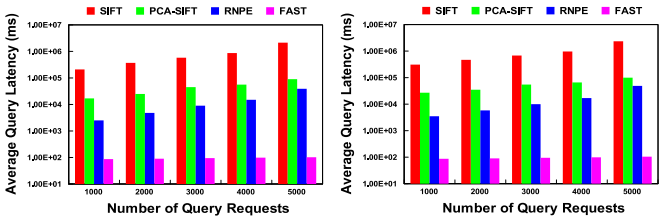
The performance of FAST is associated with its parameter settings. One of the key parameters is the metric R that regulates the measure of approximate membership. The LSH-based structures can work well if R is roughly equal to the distance between the queried point q and its nearest neighbors. Unfortunately, identifying an optimal R value is a non-trivial task due to the uncertainties and probabilistic properties of LSH [12], [36]. 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 [37] and has been used in practical applications [30], [38]. 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.

4.2 Results and Analysis

4.2.1 Query Latency

Fig. 3 shows the average query latency. The query latency includes the computation time of descriptors, e.g., image gradients and SIFT, as described in Section 3.3. We examine query performance as a function of the number of simultaneous requests from 1,000 to 5,000 with an increment of 1,000. The latency of PCA-SIFT, at 2 min, is one order of magnitude better than SIFT's 35.8 min, 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 1,000) but its performance degrades noticeably, as the number of query requests increases, to as long as 55 s. This is because the high-complexity MNPG identification algorithm and the R-tree based $O(\log n)$ query complexity of RNPE [39]. The query latency of FAST is much shorter than any of the other schemes and remains roughly at 102.6 ms 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*.



(a) Wuhan Dataset. (b) Shanghai Dataset.

Fig. 3. The average query latency.

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.

4.2.2 Query Accuracy

Table 4 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 percent 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 percent on average. The accuracy of FAST is around 99.995 percent, 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 percent 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.

4.2.3 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 Fig. 4. 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 versus 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.

TABLE 4
Query Accuracy Normalized to SIFT

Dataset	Number of Queries	SIFT	PCA-SIFT	RNPE	FAST
Wuhan	1,000	100%	99.9995%	97.3%	99.999%
	2,000	100%	99.9992%	96.5%	99.997%
	3,000	100%	99.9984%	95.9%	99.995%
	4,000	100%	99.9977%	94.1%	99.994%
	5,000	100%	99.9965%	93.5%	99.990%
Shanghai	1,000	100%	99.9992%	96.3%	99.998%
	2,000	100%	99.9988%	95.3%	99.994%
	3,000	100%	99.9982%	94.2%	99.991%
	4,000	100%	99.9969%	93.5%	99.988%
	5,000	100%	99.9957%	92.5%	99.986%

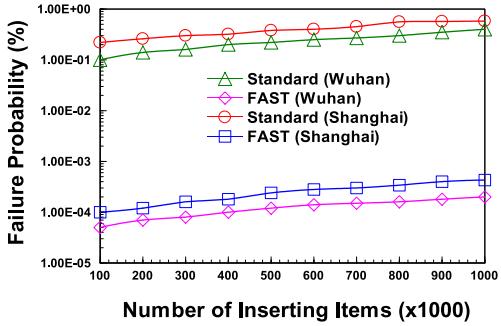
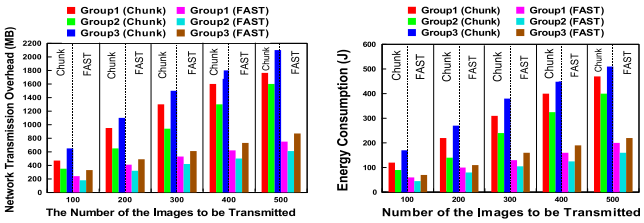


Fig. 4. Insertion failure (rehash) probability.

4.2.4 User Experiences from Smartphones

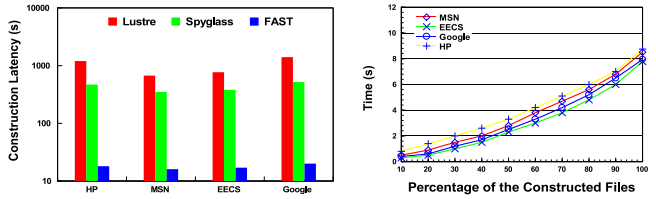
In FAST's Android-based clients, we designed a friendly and easy-to-use interface for users to upload images and submit queries. To support local image processing, we ported an open-source implementation of PCA-SIFT feature extraction algorithm to Android. Moreover, in order to comprehensively evaluate the performance, we divide 1,000 users who use this client in their smartphones into three 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 Fig. 5. 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 [40].

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. Fig. 5a shows the network transmission overhead by examining the practical use of bandwidth in transmitting a batch of images.



(a) Network transmission overhead. (b) Energy consumption.

Fig. 5. User Experiences from Smartphones.



(a) Initial Construction Time. (b) Times for different amounts of the constructed files.

Fig. 6. Time overhead in the FAST namespace construction.

We have two observations from the results. First, compared with chunk-based transmission scheme, FAST can achieve more than 55.2 percent 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 [41] 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 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 Wi-Fi.

Fig. 5b 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 percent 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.

4.2.5 Semantic-Aware Namespace

We examine the performance of FAST in terms of namespace construction time, space overhead, renaming cost and multi-node scalability. To facilitate fair and comprehensive comparisons, we use four representative system-level traces, of which three collected from the industry and one from the academia. These traces include HP file system trace [24], MSN trace [25], EECS NFS server (EECS) trace at Harvard [26] and Google clusters trace [27], which drive the FAST performance evaluation. Moreover, for the sizes of these traces, Google cluster contains 75 five-minute reporting intervals. There are a total of 3,535,029 observations, 9,218 unique jobs and 176,580 unique tasks. HP contains 94.7 million requests for a total of 4 million files from 32 users. MSN has 1.25 million files and records 4.47 million operations, in which there are 3.3 million read and 1.17 million write operations. EECS contains 4.44 million operations. The number and size of read operations are respectively 0.46 million and 5.1 GB. Those of write operations are respectively 0.667 million and 9.1 GB.

Namespace construction time. In order to construct the namespace, FAST uses locality-aware hash computation. Initially, we first serve for the in-memory files and need to construct their semantic-aware per-file namespaces. We execute these operations until a new file is loaded into the main memory. A simple LRU policy is used to handle the case that the memory is full. We then replace the namespace of a file, which is flushed to the secondary memory.

We compare FAST with Lustre [42] and Spyglass [22] in terms of the times of the initial namespace construction as shown in Fig. 6a. We observe that FAST consumes the smallest time due to its simple and low-complexity hashing computation. Lustre relies on the hierarchical tree based structure for directory indexing, thus causing much larger latency. Due to the use of K-D tree [43], Spyglass alleviates the time overhead. Fig. 6b further shows the namespace construction time with the growth of the constructed files. FAST is able to efficiently carry out the operations of namespace construction.

Space overhead. Table 5 shows the actual space overheads in FAST, Lustre and Spyglass. Lustre consumes the largest space overhead, which is derived from a large hierarchical tree for the naming service. Compared with Lustre, Spyglass uses the lightweight binary tree to reduce the space overhead. Unlike them, FAST obtains more than 60 percent space savings due to the use of simple, flat and semantic-aware namespace.

Renaming cost. In the context of FAST's namespace, the renaming operation is to change the name of a file. In order to rename a file, we need to first identify this file and then modify the file name. In general, the renaming operation also needs to update the files' metadata [44], [45].

We use a scaled-up method to intensify the used trace in terms of both spatial and temporal scales. This method has been successfully used in Glance [46] and SmartStore [23]. We first decomposed the trace into sub-traces and further add a unique sub-trace ID to all files. The start times of all sub-traces are set to zero and thus can be replayed concurrently. We maintain the chronological order among all requests within a sub-trace. The

TABLE 5
The Space Overhead

	HP	MSN	EECS	Google
Lustre	58.72 GB	15.81 GB	36.93 GB	61.52 GB
Spyglass	37.26 GB	12.53 GB	25.68 GB	47.59 GB
FAST	16.51 GB	4.75 GB	11.83 GB	19.16 GB

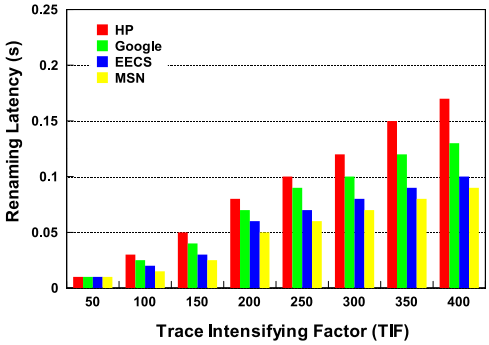


Fig. 7. Renaming latency.

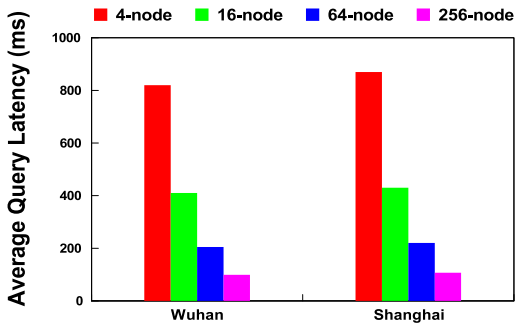


Fig. 8. Multi-node based query latency.

combined trace contains the identical histogram of file system calls like the original one but exhibits much heavier workloads. The number of sub-traces replayed concurrently is represented as *Trace Intensifying Factor* (TIF). Fig. 7 shows the latency of a "rename" operation in FAST. The average renaming latency in FAST is 0.072 s, 0.047 s, 0.052 s and 0.061 s for HP, MSN, EECS and Google traces, respectively. Due to the use of semantic grouping in $Member(f)$ and $Namespace_t(f)$ sets, FAST allows the *rename* operation to complete in one or a very small number of tuples of correlated files.

4.2.6 Multi-Node Scalability

FAST has the salient feature of efficiently supporting parallel query operations via its flat-structured addressing. This design allows FAST execution to contain a large amount of parallel queries. Fig. 8 shows the latency of queries carried out on a multi-node based system as a function of the number of nodes. We observe that the query latency decreases almost linearly with the increase in the number of nodes. Due to the promising advantage in the linear speedup, we argue that the FAST has the scalability in the multi-node system.

5 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 [47] is proposed, which includes an analysis engine for extracting instances of user-specified behavior models. ISABELA-QA [48] is a parallel query processing engine that is designed and optimized for analyzing and processing spatiotemporal, multivariate scientific data. MixApart [49] 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 [50], 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 [46], a just-in-time sampling-based system, can provide accurate answers for aggregate and top-k queries without prior knowledge. SmartStore [23] uses latent semantic indexing (LSI) tool [51], [52] to aggregate semantically correlated files into groups and support complex queries. Ceph [53] and its demonstration system [54] use dynamic subtree partition to avoid metadata-access hot spots and support filename-based query. FastQuery [55] 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 [56] proposes a locality-aware and space-efficient data structure that can efficiently support the in-memory computing. SciHadoop [57] 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 [58] proposes the idea of exploiting the backup-stream locality to reduce network bandwidth and accesses to on-disk index. Extreme Binning [59] exploits the file similarity for deduplication and can be applied to non-traditional backup workloads with low-locality (e.g., incremental backup). ChunkStash [32] maintains the chunk fingerprints in an SSD instead of a hard disk to accelerate the lookups. SiLo [60] 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 [61] 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 [62] 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 [63] is shown to be a low-bandwidth network file system. The potential of data deduplication in HPC centers is presented in [64] via quantitative analysis on the potential for capacity reduction for 4 data

centers. In order to opportunistically leverage resources on end hosts, EndRE [65] 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.

6 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.

Acknowledgments

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, and US National Science Foundation under Grants NSF-CNS-1016609 and NSF-CNS-1116606. This is an extended version of our manuscript published in the Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC), November 2014, Pages: 754-765. Yu Hua is the corresponding author.

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," *Commun. 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," in *Proc. 22nd Int. Symp. High-Perform. Parallel Distrib. Comput.*, 2013, pp. 239–250.

[3] P. Nath, B. Urgaonkar, and A. Sivasubramaniam, "Evaluating the usefulness of content addressable storage for high-performance data intensive applications," in *Proc. 17th Int. Symp. High-Perform. Parallel Distrib. Comput.*, 2008, pp. 35–44.

[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*, Dec. 2012.

[7] Y. Hua, W. He, X. Liu, and D. Feng, "SmartEye: Real-time and efficient cloud image sharing for disaster environments," in *Proc. INFOCOM*, 2015, pp. 1616–1624.

[8] Y. Ke and R. Sukthankar, "PCA-SIFT: A more distinctive representation for local image descriptors," in *Proc. IEEE Conf. Comput. Vis. Pattern Recog.*, 2004, pp. 506–513.

[9] Y. Ke, R. Sukthankar, and L. Huston, "Efficient near-duplicate detection and sub-image retrieval," in *Proc. ACM Multimedia*, 2004, pp. 869–876.

[10] J. Liu, Z. Huang, H. T. Shen, H. Cheng, and Y. Chen, "Presenting diverse location views with real-time near-duplicate photo elimination," in *Proc. 29th Int. Conf. Data Eng.*, 2013, pp. 505–56.

[11] D. Zhan, H. Jiang, and S. C. Seth, "CLU: Co-optimizing locality and utility in thread-aware capacity management for shared last level caches," *IEEE Trans. Comput.*, vol. 63, no. 7, pp. 1656–1667, Jul. 2014.

[12] P. Indyk and R. Motwani, "Approximate nearest neighbors: towards removing the curse of dimensionality," in *Proc. 13th Annu. ACM Symp. Theory Comput.*, 1998, pp. 604–613.

[13] R. Pagh and F. Rodler, "Cuckoo hashing," in *Proc. Eur. Symp. Algorithms*, 2001, pp. 121–133.

[14] Y. Hua, H. Jiang, Y. Zhu, D. Feng, and L. Xu, "SANE: Semantic-aware namespace in ultra-large-scale file systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 25, no. 5, pp. 1328–1338, May 2014.

[15] (2011). Changewave research [Online]. Available: http://www.changewaveresearch.com

[16] Science Staff, "Dealing with data-Challenges and opportunities," *Science*, vol. 331, no. 6018, pp. 692–693, 2011.

[17] X. Tan, S. Chen, Z.-H. Zhou, and F. Zhang, "Face recognition from a single image per person: A survey," *Pattern Recog.*, vol. 39, no. 9, pp. 1725–1745, 2006.

[18] T. Ahonen, A. Hadid, and M. Pietikainen, "Face description with local binary patterns: Application to face recognition," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 28, no. 12, pp. 2037–2041, Dec. 2006.

[19] X. Tan and B. Triggs, "Enhanced local texture feature sets for face recognition under difficult lighting conditions," *IEEE Trans. Image Process.*, vol. 19, no. 6, pp. 1635–1650, Jun. 2010.

[20] J. Wright, A. Y. Yang, A. Ganesh, S. S. Sastry, and Y. Ma, "Robust face recognition via sparse representation," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 31, no. 2, pp. 210–227, Feb. 2009.

[21] Y. Hua and X. Liu, "Scheduling heterogeneous flows with delay-aware deduplication for avionics applications," *IEEE Trans. Parallel. Distrib. Syst.*, vol. 23, no. 9, pp. 1790–1802, Sep. 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," in *Proc. 7th USENIX Conf. File Storage Technol.*, 2009, pp. 153–166.

[23] 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," in *Proc. Int. Conf. High Perform. Comput. Netw. Storage Anal.*, 2009, pp. 1–12.

[24] E. Riedel, M. Kallahalla, and R. Swaminathan, "A framework for evaluating storage system security," in *Proc. USENIX Conf. File Storage Technol.*, 2002, pp. 15–30.

[25] S. Kavalanekar, B. Worthington, Q. Zhang, and V. Sharda, "Characterization of storage workload traces from production Windows servers," in *Proc. IEEE Int. Symp. Workload Characterization*, 2008, pp. 119–128.

[26] D. Ellard, J. Ledlie, P. Malkani, and M. Seltzer, "Passive NFS tracing of email and research workloads," in *Proc. USENIX Conf. File Storage Technol.*, 2003, pp. 203–216.

[27] J. L. Hellerstein. (2010, Jan). Google cluster data [Online]. Available: http://googleresearch. blogspot.com/2010/01/google-cluster-data.html

[28] D. Lowe, "Distinctive image features from scale-invariant keypoints," *Int. J. Comput. Vis.*, vol. 60, no. 2, pp. 91–110, 2004.

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

[30] A. Andoni and P. Indyk, "Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions," *Commun. ACM*, vol. 51, no. 1, pp. 117–122, 2008.

[31] Q. Lv, W. Josephson, Z. Wang, M. Charikar, and K. Li, "Multi-probe LSH: Efficient indexing for high-dimensional similarity search," in *Proc. 33rd Int. Conf. Very Large Data Bases*, 2007, pp. 950–961.

[32] B. Debnath, S. Sengupta, and J. Li, "ChunkStash: Speeding up inline storage deduplication using flash memory," in *Proc. USENIX Conf. USENIX Annu. Tech. Conf.*, 2010, p. 16.

[33] FUSE [Online]. Available: http://fuse.sourceforge.net/

[34] Y. Hua, H. Jiang, and D. Feng, "FAST: Near real-time searchable data analytics for the cloud," in *Proc. Int. Conf. High Perform. Comput. Netw. Storage Anal.*, 2014, pp. 754–765.

[35] D. Lowe, "Object recognition from local scale-invariant features," in *Proc. 7th IEEE Int. Conf. Comput. Vis.*, 1999, pp. 1150–1157.

[36] A. Gionis, P. Indyk, and R. Motwani, "Similarity search in high dimensions via hashing," in *Proc. 25th Int. Conf. Very Large Data Bases*, 1999, pp. 518–529.

[37] M. Datar, N. Immorlica, P. Indyk, and V. Mirrokni, "Locality-sensitive hashing scheme based on p-stable distributions," in *Proc. 20th Ann. Symp. Comput. Geometry*, 2004, pp. 253–262.
[38] Y. Tao, K. Yi, C. Sheng, and P. Kalnis, "Quality and efficiency in high-dimensional nearest neighbor search," in *Proc. SIGMOD Int. Conf. Manage. Data*, 2009, pp. 563–576.
[39] A. Guttman, "R-trees: A dynamic index structure for spatial searching," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 1984, pp. 47–57.
[40] 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. 19th ACM Int. Conf. Multimedia*, 2011, pp. 473–482.
[41] (2012). Monsoon power monitor [Online]. Available: http://www.msoon.com
[42] Lustre [Online]. Available: http://lustre.org/
[43] J. L. Bentley, "Multidimensional binary search trees used for associative searching," *Commun. ACM*, vol. 18, no. 9, pp. 509–517, 1975.
[44] S. Ghemawat, H. Gobioff, and S. Leung, "The Google file system," in *Proc. 19th ACM Symp. Operating Syst. Principles*, 2003, pp. 29–43.
[45] K. Muniswamy-Reddy, C. Wright, A. Himmer, and E. Zadok, "A versatile and user-oriented versioning file system," in *Proc. 3rd USENIX Conf. File Storage Technol.*, 2004, pp. 115–128.
[46] H. Huang, N. Zhang, W. Wang, G. Das, and A. Szalay, "Just-in-time analytics on large file systems," in *Proc. 3rd USENIX Conf. File Storage Technol.*, 2011, p. 16.
[47] A. Viswanathan, A. Hussain, J. Mirkovic, S. Schwab, and J. Wroclawski, "A semantic framework for data analysis in networked systems," in *Proc. 8th USENIX Conf. Netw. Syst. Design Implementation*, 2011, pp. 127–140.
[48] S. Lakshminarasimhan, J. Jenkins, I. Arkatkar, Z. Gong, H. Kolla, S.-H. Ku, S. Ethier, J. Chen, C. S. Chang, S. Klasky, R. Latham, R. Ross, and N. F. Samatova, "ISABELA-QA: Query-driven analytics with ISABELA-compressed extreme-scale scientific data," in *Proc. Int. Conf. High Perform. Comput., Netw., Storage Anal.*, 2011, pp. 1–11.
[49] M. Mihailescu, G. Soundararajan, and C. Amza, "MixApart: Decoupled analytics for shared storage systems," in *Proc. 3rd USENIX Conf. File Storage Technol.*, 2013, pp. 133–146.
[50] J. C. Bennett, H. Abbasi, P.-T. Bremer, R. Grout, A. Gyulassy, T. Jin, S. Klasky, H. Kolla, M. Parashar, V. Pascucci, P. Pebay, D. Thompson, H. Yu, F. Zhang, and J. Chen, "Combining in-situ and in-transit processing to enable extreme-scale scientific analysis," in *Proc. Int. Conf. High Perform. Comput., Netw., Storage Anal.*, 2012, p. 49.
[51] S. Deerwester, S. Dumas, G. Furnas, T. Landauer, and R. Harsman, "Indexing by latent semantic analysis," *J. Amer. Soc. Inf. Sci.*, vol. 41, pp. 391–407, 1990.
[52] C. Papadimitriou, P. Raghavan, H. Tamaki, and S. Vempala, "Latent semantic indexing: A probabilistic analysis," *J. Comput. Syst. Sci.*, vol. 61, no. 2, pp. 217–235, 2000.
[53] S. Weil, S. A. Brandt, E. L. Miller, D. D. E. Long, and C. Maltzahn, "Ceph: A scalable, high-performance distributed file system," in *Proc. 7th USENIX Conf. Syst. Design Implementation*, 2006, pp. 307–320.
[54] 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: USENIX Mag.*, vol. 35, no. 4, pp. 38–49, Aug. 2010.
[55] J. Chou, K. Wu, O. Rubel, M. Howison, J. Qiang, Prabhat, B. Austin, E. W. Bethel, R. D. Ryne, and A. Shoshani, "Parallel index and query for large scale data analysis," in *Proc. Int. Conf. High Perform. Comput., Netw., Storage Anal.*, 2011, p. 30.
[56] Y. Hua, B. Xiao, B. Veeravalli, and D. Feng, "Locality-sensitive bloom filter for approximate membership query," *IEEE Trans. Comput.*, vol. 61, no. 6, pp. 817–830, Jun. 2012.
[57] J. B. Buck, N. Watkins, J. LeFevre, K. Ioannidou, C. Maltzahn, N. Polyzotis, and S. Brandt, "SciHadoop: Array-based query processing in hadoop," in *Proc. Int. Conf. High Perform. Comput., Netw., Storage Anal.*, 2011, p. 66.
[58] B. Zhu, K. Li, and H. Patterson, "Avoiding the disk bottleneck in the data domain deduplication file system," in *Proc. 7th USENIX Conf. File Storage Technol.*, 2008, p. 18.
[59] D. Bhagwat, K. Eshghi, D. Long, and M. Lillibridge, "Extreme binning: Scalable, parallel deduplication for chunk-based file backup," in *Proc. IEEE Symp. Model., Anal. Simul. Comput. Telecommun. Syst.*, 2009, pp. 1–9.
[60] 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," in *Proc. USENIX Conf. USENIX Annu. Tech. Conf.*, 2011, pp. 26–28.
[61] W. Dong, F. Douglis, K. Li, H. Patterson, S. Reddy, and P. Shilane, "Tradeoffs in scalable data routing for deduplication clusters," in *Proc. 9th USENIX Conf. File Storage Technol.*, 2011, p. 2.
[62] M. Lillibridge, K. Eshghi, D. Bhagwat, V. Deolalikar, G. Trezise, and P. Camble, "Sparse indexing: Large scale, inline deduplication using sampling and locality," in *Proc. 7th USENIX Conf. File Storage Technol.*, 2009, pp. 111–123.
[63] A. Muthitacharoen, B. Chen, and D. Mazieres, "A low-bandwidth network file system," in *Proc. 18th ACM Symp. Operating Syst. Principles*, 2001, pp. 174–187.
[64] D. Meister, J. Kaiser, A. Brinkmann, T. Cortes, M. Kuhn, and J. Kunkel, "A study on data deduplication in HPC storage systems," in *Proc. Int. Conf. High Perform. Comput., Netw., Storage Anal.*, 2012, p. 7.
[65] 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," in *Proc. 7th USENIX Conf. Netw. Syst. Design Implementation*, 2010, p. 28.

Yu Hua received the BE and PhD degrees in computer science from the Wuhan University, China, in 2001 and 2005, respectively. He is currently an associate professor at the Huazhong University of Science and Technology, China. His research interests include computer architecture, cloud computing, network storage, and cyber-physical systems. He has more than 60 papers to his credit in major journals and international conferences including *IEEE Transactions on Computers (TC)*, *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, USENIX ATC, USENIX FAST, INFOCOM, SC, ICDCS, ICPP, and MASCOTS. He has been on the organizing and program committees of multiple international conferences, including INFOCOM, ICDCS, ICPP, RTSS, and IWQoS. He is a senior member of the IEEE and CCF, a member of the ACM, and USENIX.

Hong Jiang received the BSc degree in computer engineering from the Huazhong University of Science and Technology, Wuhan, China, in 1982, the MASc degree in computer engineering from the University of Toronto, Toronto, Canada, in 1987, and the PhD degree in computer science from the Texas A&M University, College Station, in 1991. Since August 1991, he has been at the University of Nebraska-Lincoln (UNL), where he served as a vice chair of the Department of Computer Science and Engineering (CSE) from 2001 to 2007, and is a professor of CSE. His present research interests include computer architecture, computer storage systems, and parallel/distributed computing. He serves as an associate editor of the *IEEE Transactions on Parallel and Distributed Systems*. He has more than 180 publications in major journals and international conferences in these areas. He is a fellow of the IEEE and a member of the ACM and ACM SIGARCH.

Dan Feng received the BE, ME, and PhD degrees in computer science and technology from the Huazhong University of Science and Technology (HUST), China, in 1991, 1994, and 1997, respectively. She is a professor and vice dean of the School of Computer Science and Technology, HUST. Her research interests include computer architecture, massive storage systems, and parallel file systems. She has more than 80 publications to her credit in journals and international conferences. She is a member of the IEEE.