Journal of Network and Computer Applications 192 (2021) 103167



Contents lists available at ScienceDirect

Journal of Network and Computer Applications

journal homepage: www.elsevier.com/locate/jnca





Strategizing secured image storing and efficient image retrieval through a new cloud framework

Jannatun Noor [a,b], Saiful Islam Salim [a,*], A.B.M. Alim Al Islam [a]

[a] Department of Computer Science and Engineering, Bangladesh University of Engineering and Technology, Dhaka, Bangladesh
[b] Department of Computer Science and Engineering, BRAC University, Dhaka, Bangladesh

ARTICLE INFO

Keywords:
Image cloud
OpenStack Swift
PJPEG
Middleware
Image resizing
Encryption

ABSTRACT

Nowadays, a major problem in faster page loading through optimizing websites is the absence of having images in their intended sizes. Accordingly, while loading a webpage, most of the time overhead pertains to image-related tasks such as image loading and resizing, which can be optimized substantially through the pre-availability of smaller-size images. Therefore, in this study, we propose a strategy to enable faster and efficient image retrieval from a cloud via the necessary pre-processing of images beyond conventional online processing. We also extend the abilities of cloud file sharing from the conventional "only storing images" to pre-processing along with security reinforcing. Here, we perform the first task of resizing images to several dimensions for covering diversified remote user devices available nowadays. Then, we perform encoding and encryption of images using the P-Fibonacci transform of Discrete Cosine Coefficients (PFCC) algorithm. Afterward, we resize images using the Bicubic interpolation method to both JPEG and Progressive-JPEG (PJPEG) formats by adding a new middleware named iBuck for faster and smooth retrieval. We leverage necessary algorithms and image types after careful evaluation of their performances in our intended framework. Our evaluation covers both objective and subjective evaluations including QoE metrics such as MSE, SSIM, etc. Furthermore, we conduct detailed experimentation over real setups to evaluate the performance of our implemented middleware in a cloud file sharing environment with both custom and standard data sets. Our evaluation reveals substantial performance improvement using our proposed framework compared to that using conventional alternatives.

1. Introduction

The demands of secured data storage, faster data retrieval, and secured communication from the client end to server end — are the major concerns nowadays. This happens as online connectivity is getting exponentially increased through communicating with diversified connected devices in recent times. Over the online domains, users are now producing and consuming more "unstructured" data than ever with online videos, social media such as Facebook, Twitter, etc., gaming, user-uploaded contents, and Software-as-a-Service applications — all contributing to the vast need for easily accessible storage systems that can grow without bounds. One of the most contributing parts in this stored data comprises multimedia images. In this regard, the advancement of cloud computing presents it as a widely accepted solution to store multimedia images. Not only that, owing to the immense computing power generally available in the cloud architectures, cloud computing is now also being extensively used for processing images.

1.1. Image processing and aspects of its execution on cloud

In our day-to-day life, images have substantial value for diversified purposes including sensitive applications such as medical diagnosis. For example, in medical imaging and radiology, researchers are now acquiring large data sets that can be raw in formats. These images can vary significantly in their sizes and can consume huge storage. Aligning images, detecting an object in an image via computer vision algorithms, extracting a region of interest from an image and transforming it to a standardized format for display, and finding similar images by analyzing their features, etc., are some common computations and analytics generally done on the images. In most cases, fetching large images from public storage systems to own processing systems and then processing those images in the own processing systems — both appear to be expensive and time-consuming.

* Corresponding author at: Department of Computer Science and Engineering, Bangladesh University of Engineering and Technology, Dhaka, Bangladesh.
E-mail addresses: jannatun.noor@bracu.ac.bd (J. Noor), 1018052067@grad.cse.buet.ac.bd (S.I. Salim), alim_razi@cse.buet.ac.bd (A.B.M.A.A. Islam).

https://doi.org/10.1016/j.jnca.2021.103167
Received 24 January 2021; Received in revised form 8 July 2021; Accepted 23 July 2021
Available online 29 July 2021
1084-8045/© 2021 Published by Elsevier Ltd.

Besides, having the images during communication and in public storage systems may often open doors for malicious users to perform activities that are not legitimate. Consequently, storing images in public clouds can significantly increase retrieval delay along with making them vulnerable to different well-known security threats. Examples of such contemporary and future security threats (Khan, 2016; Chidambaram et al., 2020; Devaraj et al., 2020; Tak et al., 2017; Hunt et al., 2020) include untrusted execution environment (Futral and Greene, 2013), protocol vulnerability (Gruschka and Iacono, 2009; Kumar and Goyal, 2019; Li et al., 2019), etc.

A possible solution to ensuring security is to have its private cloud, which can provide image security as well as high availability of those images at any end. However, to the best of our knowledge, there is still no private cloud for maintaining images securely while making them highly available for any type of mobile or web client. Besides, if anyone wants to access important images anywhere using any type of remote device, then the cost of transformations and resizing of images come into play. Here, if we perform computation needed for transformations and resizing on the fly, then processing power and delay can severely increase.

1.2. Existing studies on image processing over cloud

Several existing studies (Lahan, 2021; Sheidani et al., 2021) mainly focus on only image processing operations covering resizing, cropping, rotating, compressing, encrypting, etc. Moreover, these studies investigate these features on a public cloud. Besides, all the operations are done on the fly. Thus, these studies are yet to focus on offline image preprocessing while storing them in the cloud. In addition, these studies do not focus on how to build a private image cloud storage system.

Another study (Torres et al., 2018) investigates encryption of images stored in the cloud and execution of user-defined programs inside an object storage cloud in the big data context. The study presents a new image encryption algorithm namely "PFCC". However, it does not focus on any kind of image resizing (either online or offline) for faster image retrieval.

Many other algorithms have been proposed for securing digital images (Lian et al., 2007; Wu and Kuo, 2005; Lin et al., 2005; Zeng and Lei, 2003) since the mid-1990s. These studies cover several aspects such as encryption (Jia et al., 2011), confidentiality (Rad et al., 2015), etc. These classical security algorithms are for general purposes and are not specialized for cloud framework.

Only a few research studies (Doelitzscher et al., 2011; Dong et al., 2009; Dong et al., 2009) present operating with own private cloud infrastructure. However, these studies are for special purposes such as e-Learning. Thus, image processing and enhancing the security of the images are yet to be considered in these types of existing studies.

There remain numerous research studies in the literature that deal with general-purpose operations on the cloud. Examples include data recovery (Jia et al., 2011), virtual machine mapping (Lundh and Clark, 2021), data security, etc. However, all such studies are for generic data having no consideration of specialized processing of images. Moreover, these studies are intended for public clouds, and thus, they do not realize the nature of exclusive usage as can be performed in private clouds.

1.3. Motivations and implications of our study

Our study is motivated by the limitations of state-of-the-art research studies and the challenges involved in developing a private cloud storage system, specifically designed for secured image processing. In this regard, a baseline version of image cloud architecture is presented in Noor and Al Islam (2017) using a middleware named *'iBuck'*. Accordingly, in this study, we propose an extended architecture of private image cloud exploiting the updated middleware named *'iBuck'*, which stands for *image Bucket*.

Here, we develop different Python-based image processing and resizing modules for implementing the middleware. While exploiting different resizing algorithms offline, we integrate our developed modules in an OpenStack object service named Swift (Arnold, 2015). OpenStack is widely used for auditing virtual networks isolation due to its dynamic and layered nature (Madi et al., 2018). Here, users can upload images, which will be pre-processed through resizing and encrypting according to several dimensions of potential remote devices. There are different sensitive applications of our proposed architecture. Examples include medical imaging systems, military image communication, mobile commerce, etc.

1.4. Our contributions

Based on our work, we make the following set of contributions in this study:

- We propose a new framework for secured image processing in a private cloud. The architecture enables a novel mode of moving and executing user-defined programs for resizing and encryption–decryption of images near the data inside of an object storage cloud.
- We integrate resizing and encryption–decryption algorithms as a secured proxy service combined with a cloud file sharing environment named Swift, which is an OpenStack object service. We perform necessary changes in the software to enable resizing and encryption–decryption tasks.
- We experimentally evaluate the performance of our proposed architecture in a real private cloud setup. Our experimentation covers both local and remote servers using both custom and standard data sets. In experimentation, we perform necessary parameter tuning. Besides, we perform both subjective and system-level evaluations to demonstrate the efficacy of our proposed framework.

1.5. Organization and extensions made in this study

In course of the enhancement, in this study, we newly analyze relevant studies on Software-Defined Storage (SDS) (Section 2.1), OpenStack Swift (Section 2.2), image resizing techniques (Section 2.3.1), progressive JPEG (Section 2.4), image encryption–decryption techniques (Section 2.4.1), Quality of Experience (QoE) (Section 2.5), and other related work on cloud framework (Section 3). We present newly-designed methodologies of resizing images to several dimensions (Fig. 4), creating progressive JPEG images (Section 4.2), encryption and decryption of images (Section 4.3), deploying a new middleware in OpenStack Swift (Section 4.4), and request analysis of the proposed architecture (Section 4.5). Further, we perform new experimentation, which we present throughout Section 5. Limitations of our proposed system are also presented in Section 6. Besides, our work also opens several future research avenues, which are presented in Section 7.

2. Background

Unstructured data usually demands to be stored in such a manner that guarantees persistence, availability, and manageability — all at an economical expense. OpenStack Swift (Arnold, 2015) can exhibit all the fundamental characteristics of a cloud storage system in this regard. Hence, we choose this platform and attempt to modify the system for deploying an image storage system. Here, our focus especially pertains to private cloud storage considering its applicability. Therefore, we present the necessary background information in the following sections.

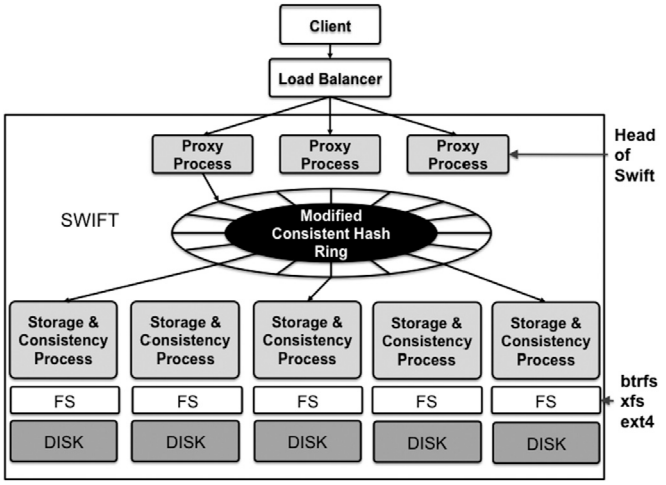


Fig. 1. Overview of Swift architecture (Noor et al., 2017).

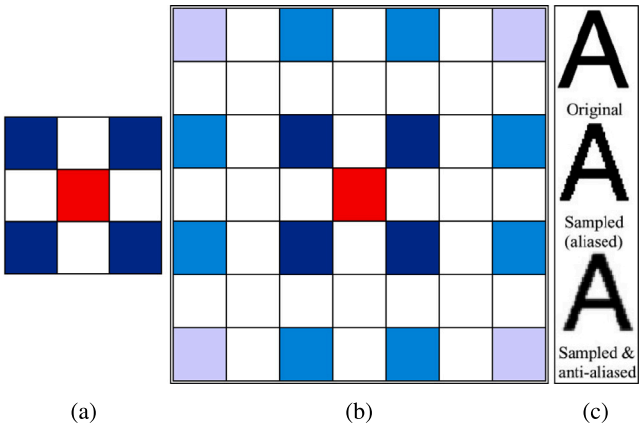


Fig. 2. Image resizing techniques: (a) Bilinear interpolation, (b) Bicubic Interpolation, (c) Outcome without and with Antialiasing.

2.1. Software-Defined Storage (SDS)

In recent times, unstructured data storage is forcing a standard change in storage architecture due to the scale of it, hence Software-Defined Storage (SDS) enters the story. With SDS, the entire storage stack has regained the criteria of persistence, availability, low cost, manageability, and quick failure recovery on standard and open-server hardware. An SDS system distributes the intelligence and access from the underlying physical hardware. SDS system has four key components — Storage routing, Storage resilience, Physical hardware, and Out-of-band controller (Arnold, 2015).

Without any downtime and with no demand for physical data migration in an SDS system, capacity management, up-gradations, expansions, and decommissions can be accomplished. The detachment of physical hardware from the software enables mix-and-match hardware configurations within the corresponding storage system and just-in-time purchasing. As being open-source, SDS encourages an extensive growing ecosystem, where the diversity of the community members encourages standards and tools.

2.2. OpenStack Swift

OpenStack Swift (Arnold, 2015) is an open-source object storage system — designed and developed based on SDS, which allows achieving high availability, redundancy, throughput, and capacity. It is competent to store billions of objects distributed over several nodes, and thus, is used to store any number of large and small objects by employing its RESTful HTTP API. A Proxy Server is accountable for handling metadata. Here, for each GET or PUT request, the server search for the locations of the account, container, or data object in the ring and routes the request accordingly. Storage servers are principally Account, Container, and Object servers. The ring depicts a mapping between the names of entities stored on disk and their physical locations (Arnold, 2015). OpenStack Swift empowers users to store unstructured data objects with a canonical name comprising three components: account, container, and object. Using one or more of these components enables the system to generate a unique storage location for data.

Users can access the storage system through the native HTTP interface or command-line tools, file system gateways, or easy-to-use applications to store and synchronize data with their desktops, tablets, and mobile devices. Swift has no transaction or locking delays focusing on availability over consistency. Large numbers of simultaneous reads, as well as writes, are fast to accomplish. In addition to the components, Swift has several consistency processes such as auditor, replicator, account reaper, container, object updater, and object expirer. Different operations such as PUT, GET, DELETE can be accomplished utilizing these components and processes of Swift. (Arnold, 2015). However, OpenStack Swift is mainly focused on object storing and retrieval of any kind of object. Hence, there are many scopes of adding image processing tasks in OpenStack Swift which is a little focused in Swift. Fig. 1 presents an overview of Swift architecture.

2.3. Image processing

Recently, due to the development of numerous advanced digital viewing equipment, there is a thriving demand for images to be resized for an appropriate view of such applications. Hence, images need to convert to several low resolutions according to cover several diversified devices by using image interpolation. Image interpolation is a method of enlargement of a digital image, i.e., producing a high-resolution image from an available low-resolution image. Image interpolation is also beneficial in computer graphics, editing, online image viewing, bio-medical domain, remote sensing, and other disciplines.

2.3.1. Image resizing techniques

The interpolation algorithms for image resizing are of two types - non-adaptive and adaptive. Certain computations are performed indiscriminately to the entire image without considering the image contents in non-adaptive algorithms, including Nearest Neighbor, Bilinear, Bicubic, Spline, Cubic Polynomial, Sinc, Lanczos, and others (Roszkowiak et al., 2017). Moreover, in adaptive algorithms, the logic employed for computing the unknown high-resolution pixels depends upon the image features. Licensed software such as Qimage, PhotoZoom Pro, Genuine Fractals includes adaptive algorithms (Kishorebabu et al., 2017).

The Nearest Neighbor algorithm chooses the nearest pixels without considering the other neighboring pixels. It results in the least computation and least processing time. However, it is not suitable to enlarge images, as it introduces undesired artifacts near the edges where we can obtain more comprehensive intensity variations and not smooth results (Roszkowiak et al., 2017). Bilinear interpolation contemplates the closest 2×2 neighborhoods of the associated pixels surrounding the unknown pixels and then exerts the weighted mean of these four pixels to get the missing pixel value (Fig. 2a).

Bicubic interpolation contemplates the closest 4×4 neighborhood of the associated pixels encompassing the unknown pixel and exerts the weighted mean of these 16 pixels to generate the interpolated pixel. More neighboring pixels are assigned higher weights since all the 16 neighboring pixels are not at equal distance from the unknown pixel (Fig. 2b). Antialiasing attempts to minimize the expression of aliased or rough diagonal edges termed 'jaggies' and provides smoother edges

with higher resolution. The aliased edge rounds up or down with no intermediate value, whereas the anti-aliased edge provides a value proportional to the percentage of the edge within each pixel. These deliver texts or images with a more digital appearance (Fig. 2c).

2.4. Progressive JPEG

JPEG-type images are considered in the most image displaying programs on the web when downloading or saving an image in 'Baseline' mode. The best approach to visualize baseline mode is that the image begins downloading or displaying at the top and is rendered line by line. By rolling down, each line of pixels displays at a time during compression and/or presentation of the image.

On the contrary, 'Progressive JPEG' displays the whole image right away. The trade-off here is that it only loads some of the image data at a time. The image looks pixelated before loading the full size and will become clearer as it loads more (Harrison et al., 2010). Fig. 6 presents the baseline and progressive JPEG type images.

2.4.1. Image encryption–decryption techniques

Security of data/images is one of the essential aspects in the domain of digital transfer. Encryption is one of the most well-known methods for protecting the confidentiality of images in a reliable and unrestricted public medium. This medium is usually vulnerable to attacks, and therefore, efficient encryption algorithms are needed to guard image data transfer.

Some of modern cryptography algorithms are Vigenere Cipher (Kumari et al., 2017), Data Encryption Standard (DES) (Kumari et al., 2017), International Data Encryption Algorithm (IDEA) (Kumari et al., 2017), Blowfish (Kumari et al., 2017), Visual Cryptography (Kumari et al., 2017), RC4 (Kumari et al., 2017), RC5 (Kumari et al., 2017), RC6 (Kumari et al., 2017), Triple Data Encryption Standard (TDES) (Kumari et al., 2017), Advanced Encryption Standard (AES) (Kumari et al., 2017), Intertwining Chaotic Maps (Kumari et al., 2017), Chaotic Function Using Linear Congruence's (Kumari et al., 2017), Mixed Transform Logistic Maps, (Kumari et al., 2017) and Chaotic Map (Kumari et al., 2017). These types of algorithms are more complex to use when batch parallel processing is needed in cloud computing and images are delivered to mobile cloud users. Several approaches (Agaian et al., 1995) are presented concerning the improvement of the security of mobile cloud users. Furthermore, they are not suitable for mobile cloud security while users leave the mobile cloud often. Hence, study (Torres et al., 2018; Noor and Al Islam, 2017; Noor et al., 2017) propose dual image encryption Proxy Service for OpenStack object storage environment to accommodate the privacy and authentication based on p-Fibonacci Transformation of Cosine Coefficients (PFCC) (Jia et al., 2011) of an image.

2.5. Quality of Experience (QoE)

For lossy image compression, developers need to find out the best quality of experiences using popular QoE techniques. Both objective and subjective factors are required to quantify users' QoE. Objective factors comprise parameters in the Network layer (jitter, packet loss, delay, etc.) and Application layer (resolution, frame rate, etc.) (Aroussi et al., 2012). However, subjective factors circumscribe the visual separation of an image from its definition, which is influenced by its resolution. Subjective factors are recognized to be more complex than objective factors. Subjective factors can also be used to incorporate users' psychological states such as preference and demographic profile such as — age, gender, etc. We present commonly used image quality metrics (Wang et al., 2017; Pal and Vanijja, 2017; Díaz et al., 2020) used for subjective and objective measurements of QoE in the following subsections.

Mean Opinion Score (MOS): MOS is taken as the scoring criterion for subjective QoE analysis, which exhibits the evaluation of some test panels (Pal and Vanijja, 2017). Additionally, MOS is utilized for the qualitative representation of an image such as 'good' or 'very bad' using a numerical method of denoting the quality.



Fig. 3. MOS outcomes depend on an user.

Table 1
MOS table for five-point scale.

MOS	Quality	Impairment
5	Excellent	Imperceptible
4	Good	Perceptible but not disturbing
3	Fair	Slightly disturbing
2	Poor	Disturbing
1	Bad	Very disturbing

MOS is represented as a number ranging from 1 to 5, where '1' refers to the worst, and '5' refers to the best experience (Fig. 3 and Table 1). However, there are software applications (Vatolin et al., 2021) that measure MOS on networks.

The minimum threshold for acceptable quality corresponds to a MOS is 3.5 (Pal and Vanijja, 2017). Due to the human tendency to avoid perfect ratings (now reflected in the objective approximations), somewhere around 4.3–4.5 is considered an excellent quality target.

Peak-Signal-to-Noise-Ratio (PSNR): PSNR provides the ratio (in dB) between the power of the original signal and the reconstructed compressed signal. PSNR is usually calculated via mean squared error (MSE) between two signals considering the maximum possible luminance of images. MSE and PSNR are computed as follows (Wang et al., 2017; Díaz et al., 2020).

$$MSE = \frac{1}{mn}\sum_{i=0}^{m-1}\sum_{j=0}^{n-1}[I(i,j) - K(i,j)]^2 \tag{1}$$

$$PSNR = 10\log_{10}\left(\frac{{MAX_I}^2}{MSE}\right) \tag{2}$$

Eqs. (1) and (2) depict MSE and PSNR of a noise-free $m \times n$ monochrome image I and its noisy approximation K. Here, MAX_I is the maximum possible pixel value of the image. The higher the value of PSNR, the better the quality of an image.

Structural Similarity Index (SSIM): SSIM is a perceptual metric that quantifies image quality degradation induced by processing such as data compression or data losses in transmission. It requires the reference image and the processed image. Higher SSIM indicates the better quality of a reference image. SSIM utilizes a structural distortion-based measurement procedure (Wang and Li, 2007). Structure and similarity in this context refer to representations of the signals having strong dependencies between each other, especially in close proximity (Wang and Bovik, 2009).

3. Related work

Traditional cryptographic algorithms/systems for data security are often not fast enough to process the vast amount of data generated by multimedia applications to meet real-time constraints. Therefore, several research studies have been performed in this regard. For example, Zeng and Lei (2003) present a joint encryption and compression framework in which video data are scrambled efficiently in the frequency domain by employing selective bit scrambling, block shuffling, and block rotation of the transform coefficients and motion vectors. It

allows scalability, and other content processing functionalities without having access to a cryptographic key.

In the domain of cryptography, Zeng and Lei (2003) group the image encryption techniques into two major categories namely non-chaos methods and chaos-based methods in their survey. Besides, Zhang et al. (2020a) propose a secure and efficient outsourcing protocol for face recognition through principal component analysis. Additionally, Roszkowiak et al. (2017) compare the advantages and disadvantages of the feature of image resizing through nearest-neighbor interpolation, bilinear interpolation, bi-cubic interpolation, and cubic B spline interpolation during magnification. Nonetheless, Tanwar et al. (2020) proposed an image symmetric encryption scheme utilizing Permutation Ordered Binary Number System (Singh et al., 2017) and pseudo-randomly generated stumble matrix that partially preserves the trichotomous property between the pixel intensity values of the input image. They also present a model named CryptoLesion, which can perform secure image segmentation tasks using Whale Optimization Algorithm (WOA) in the Encrypted Domain (ED) over the cloud.

There exist different studies on cloud computing-based applications. For example, Dong et al. (2009) propose the use of cloud computing as a base for modern e-Learning applications that can leverage cloud computing for dynamic handling of assignable storage and computing resources. Besides, another study (Doelitzscher et al., 2011) namely CloudIA performs authentication and integration with existing IT infrastructure, creation of customized on-demand virtual machines, and integration with a public cloud provider for enabling an e-Learning ecosystem in the cloud.

BlueSky cloud framework (Dong et al., 2009), having similar architectural layers as CloudIA (Doelitzscher et al., 2011), enables physical machines and allocating them on-demand for e-Learning systems. However, it does not have a security layer similar to the CloudIA architecture and only focuses on delivering IaaS, whereas the CloudIA offers IaaS, PaaS, and SaaS. The Virtual Computing Laboratory (VCL) (Vouk et al., 2009) enables students to reserve and access virtual machines (VMs) with a basic image or specific application environments, such as Matlab and Autodesk. However, the VCL does not offer collaboration features such as CollabSoft. The VCL offers IaaS and PaaS platforms, which could then be used to host collaboration systems (SaaS) on top of it.

The Snow Leopard Cloud (Cayirci et al., 2009) provides PaaS for North Atlantic Treaty Organization (NATO) to run its various military exercises and mission events. Besides, Snow Leopard Cloud is used to run Web 2.0 applications such as video teleconferencing, voice over IP, and remote management over handheld devices and terminals. As the Snow Leopard Cloud is targeted towards military usage, it has multi-level security and the network infrastructure is encrypted.

As per the current technology goes, there exist four different types of cloud models nowadays: Public, Private, Hybrid, and Community. Zhao et al. (2019) attempt to answer the question of which model would be most beneficial for business purposes. McCarthy et al. (2021) present a system and method for private cloud computing along with the development and deployment of cloud applications over a private cloud. The system and method include components such as a cloud controller, a cloud stack, a Service Registry, and a cloud application builder.

On the other hand, many existing research studies on secured and available cloud systems focus on privacy and security technologies, whereas few address the DR process particularly for a Big Data system. However, most of the recent studies that have investigated DR methods belong to the 'single-basket' approach, which means there is only one destination from which to secure the restored data, and mostly use only one type of technology implementation. In this regard, Chang (2015) propose a 'multi-purpose' approach, which allows data to be restored at multiple sites with multiple methods to ensure maximum data recovery. Such an approach is applicable to cloud systems.

Regarding enabling virtual machines over a cloud system, Ni et al. (2011) present a virtual machine mapping policy based on multi-resource load balancing. It uses resource consumption of the running virtual machine along with a self-adaptive weighted approach to resolve load balancing conflicts of each independent resource resulted from different demands for resources placed by different cloud applications. Apart from these studies, for enhancing the security of mobile cloud users, a few proposals have been presented. For example, Jia et al. (2011) design a secure mobile user-based data service mechanism (SDSM) to provide confidentiality and fine-grained access control for data stored in the cloud. Zheng et al. (2017) propose a privacy-preserving image denoising technique from external cloud databases, which enables the cloud hosting encrypted databases to provide secure query-based image denoising services. Dong et al. (2020) propose secure plaintext image storage protocols in the cloud environment for image owners to manage and control their outsourced images. Namasudra et al. (2020) propose a novel Deoxyribonucleic Acid (DNA)-based encryption scheme for securing multimedia files in the cloud computing environment.

Furthermore, to facilitate privacy-preserved image retrieval from mobile devices, Rahim et al. (2018) use a pre-trained CNN model (VGG-16) to automatically learn rich features from the user's data for avoiding the time-consuming efforts of the features engineering. Another study (Zhang et al., 2020b) proposes a triplet deep CNN hashing model named TDCNN-HASH to generate efficient and compact hash codes for images. It utilizes a triplet CNN structure that is trained by triplet labels to learn the deep visual features and deep hash code simultaneously. Such existing secure image retrieval models often face difficulties in ensuring minimum retrieval performance. To resolve this issue, a resrach study (Devaraj et al., 2020) presents a Secure Image Archival and Retrieval System (SIARS) using deep learning (DL) and multiple share creation schemes. The proposed SIARS model involves AdaGrad based Convolutional Neural Network (AG-CNN) based feature extractor to extract the useful set of features from the input images. Another study (Liu et al., 2019) uses a pre-trained deep CNN model to extract the deep features of an image. Here, the information about the neural network is strictly concealed by utilizing a lattice-based homomorphic scheme.

Finally, in the case of image retrieval, there exist several research studies. For example, Tzelepi and Tefas (2018) propose a model retraining method for Content-Based Image Retrieval (CBIR) through learning more efficient convolutional representations. Wang et al. (2019) propose a basic privacy-preserving CBIR framework to reduce the storage and communication overhead significantly through designing a new efficient key conversion protocol to support unshared key multi-owner multi-user image retrieval without losing search precision. Hassan et al. (2021) propose and implement a secure CBIR framework that performs image retrieval on the cloud without the user's interaction. They also present a pre-trained generic DNN model (e.g., VGG-16), which is used to extract the feature vectors of an image on the user side. On the other hand, the cloud servers perform secure image inference with a private pre-trained DNN model and execute Approximate Nearest Neighbor (ANN) image retrieval protocols without the user's any more interaction. Moreover, Huang et al. (2019) propose a secure image retrieval scheme that employs fine-tuned CNNs to extract semantic features and allows clouds to build a secure index graph locally to improve search speed. Such existing privacy-preserving CBIR schemes attempt to guarantee image privacy while supporting image retrieval. However, these schemes still have inherent limitations such as key leakage, low search accuracy, low search efficiency, etc., Li et al. (2020) and Xia et al. (2021).

All these existing studies are mostly relevant to the public cloud. To the best of our knowledge, the approach of using private clouds to escalate image security and investigating aspects of the private cloud to enhance its performance in this regard are yet to be focused in the literature. Hence, we propose a new private cloud framework for

enhancing image security and image processing. We develop a new middleware architecture for this purpose inside the cloud file sharing environment namely OpenStack Swift. We will show in later sections how our proposed design works. We name our prototype as iBuck. In Table 2 we show a comparison of iBuck with the existing research studies including the recent studies adopting machine learning-based systems. As clearly portrayed in the table, iBuck is different from all the other existing studies pertinent to image resizing, image conversion, cloud deployment, and security, as iBuck provides all these features in a single system covering efficient support for diversified remote devices.

It is worth mentioning that our work in this paper is an extension of our preliminary work presented in Noor and Al Islam (2017). Note that, in our prior study, we proposed an architecture of private image cloud exploiting a middleware named 'iBuck'. In that study, we developed different Python-based image processing and resizing modules for implementing the middleware. We developed modules with an OpenStack object service named Swift where users can upload images, which will be resized and encrypted according to several dimensions of remote devices. In this paper, we extend the earlier study (Noor and Al Islam, 2017) through detailing the framework along with more rigorous experimentation and analyses thereof.

4. Our proposed methodology

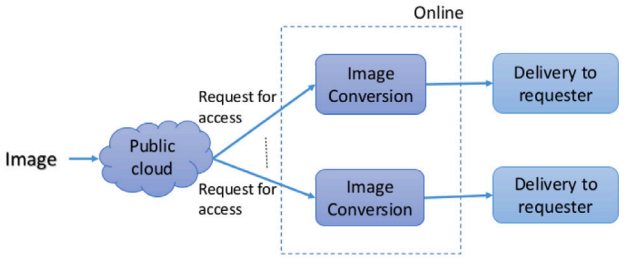
In the conventional model of an image storing and retrieval system, all types of image processing are done in an online manner. Here, when a user requests for uploading an image (PUT/POST request), the image is stored directly in the public cloud. Besides, for a download request (GET request), at first, the original version of that specific image is sent to the public cloud. Then, from the public cloud, an online processing request is sent to a specific server that performs the tasks of image conversion such as image resizing. Later the converted image is delivered to the requester. Fig. 4a presents an overview of the storing and retrieval of images through a conventional model.

On the contrary, in our proposed private cloud model, we focus on offline pre-processing for faster retrieval of images. We present our proposed model in Fig. 4b. As the figure depicts, the basic difference between our model and the conventional one is to perform offline pre-processing of images in our model compared to the processing as done in its counterpart.

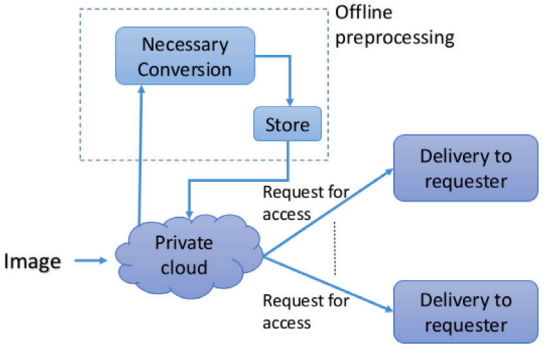
Further, in Fig. 4c, we present the detailed architecture of the necessary conversion as performed in the models. During the conversion task, there exist three key tasks to be performed in the cloud storage when uploading an image. The first task is to perform resizing of images to several dimensions to cover diversified devices. The second task is to create a progressive JPEG. Finally, we perform encryption of all the generated versions of the images.

To be specific, in an end-to-end operation in our proposed model, we perform several key steps when a user requests for uploading an image (PUT/POST request). Here, at first, the user sends a request to the proposed cloud server. In doing so, the request is first sent to the proxy server and the proxy server receives the image for processing. Then, the receiver server (Proxy) receives the image, does necessary processing tasks, and sends the several processed images to store in another server (Storage server). In the meantime, a response is sent to the requester. These steps are not visible to the users.

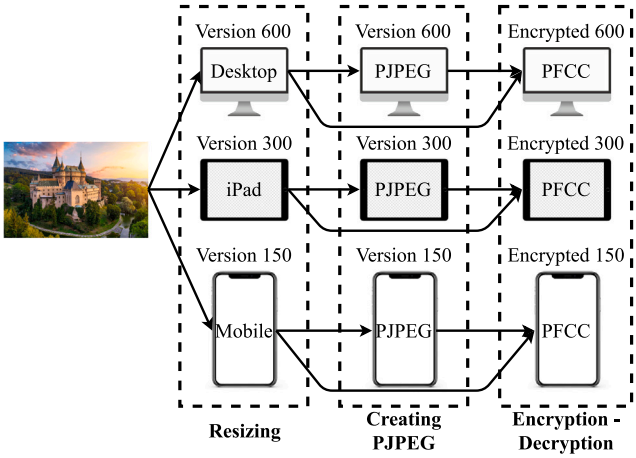
Moreover, when a user requests for downloading an image (GET request), only the specific version of the image (i.e., higher size for displaying web or desktop images, medium size for iPad, smartphone, and lower size for displaying thumb type of images) is downloaded from the cloud server. Here, no extra processing is needed for downloading an image. Hence, image retrieval is faster than conventional public cloud.[1] We present the key steps of our proposed mechanism in the following sections.



(a) Conventional model



(b) Proposed model



(c) Flow of necessary conversions in the "Online" and "Offline pre-processing" blocks presented in the above two models

Fig. 4. Conventional and proposed models for storing and retrieving images from the cloud.

4.1. Resizing of images to several dimensions

Nowadays, image quality and resolution, are very important for different types of web and mobile applications. For social images, all

[1] Conventional public cloud refers to clouds operated by the public cloud service providers such as Dropbox, Cloudinary, Sync, etc. Some of the conventional public service providers provide storage systems using native OpenStack Swift Object Storage as it is open-source.

Algorithm 1 Algorithm for resizing an raw image data based on given input dimension

```
procedure resizeImage(dm, rawImage)                                   ▷ dm is for dimension
    newWidth ← dm
    newHeight ← dm
    if (dm ≤ imageDm) then
        resizeImage ← rawImage
    else
        if (rawImage.size[0] ≤ rawImage.size[1]) then   ▷ rawImage width = rawImage.size[0], rawImage height = rawImage.size[1]
            hPercent ← (newHeight / float(rawImage.size[1]))
            newWidth ← int((float(rawImage.size[0]) * float(hPercent)))
        else
            wPercent ← (newWidth / float(rawImage.size[0]))
            newHeight ← int((float(rawImage.size[1]) * float(wPercent)))
        end if
        resizeImage ← rawImage.resize(newWidth, newHeight)   ▷ Using interpolation method (PIL.Image.BICUBIC)
    end if
    return resizeImage
end procedure
```

Algorithm 2 Algorithm to upload an image data to object storage using HTTP request

```
1:  procedure uploadImage(image, environment, path, application)   ▷ Image will be uploaded to destination path using application and environmegent
2:      if (img > MAX_FILE_SIZE) then
3:          raise uploadError
4:      end if
5:      new_envrn ← newEnv(environment)                  ▷ Built-in method of Swift to create a new environment
6:      request ← objectReq(path, new_envrn)             ▷ Built-in method of Swift to create a HTTP request
7:      res ← request.getResponse(application)           ▷ Built-in method of Swift to get HTTP response
8:      if (res ≠ SUCCESS) then
9:          raise uploadError
10:     end if
11: end procedure
```

Algorithm 3 Algorithm of iBuck middleware

```
1:  procedure iBuckApp(app)
2:      if reqMethod = GET then
3:          DecryptPFCC(origImage)                       ▷ Decryption function using PFCC
4:          return
5:      end if
6:      if original image not in right format then
7:          im.save(origImage, JPEG)                     ▷ PIL library function
8:      end if
9:      encryptImage ← EncryptPFCC(origImage)            ▷ Encryption function using PFCC
10:     uploadImage(encryptImage, env, path, app)        ▷ origImage, env, and path are global variable
11:     im.save(origImage, JPEG, progressive = True)
12:     encryptImage ← EncryptPFCC(origImage)
13:     uploadImage(encryptImage, env, path, app)
14:     for d in [resizeDimensions] do                   ▷ resizeDimensions = 600, 300, 150
15:         img ← resizeImage(d, origImage)
16:         enIm ← EncryptPFCC(img)
17:         uploadImage(enIm, env, path, app)
18:         im.save(img, JPEG, progressive = True)
19:         enIm ← EncryptPFCC(img)
20:         uploadImage(enIm, env, path, app)
21:     end for
22:     return
23: end procedure
```

types of devices can mostly be covered through three different types of resolutions of the main image — 600 for iPad or web, 300 for medium-size mobile devices, and 150 for thumb size (Fig. 5).

Hence, we apply conventional interpolation methods for resizing original images to several dimensions using interpolation techniques. Interpolation techniques can help to convert an image from different resolutions keeping almost the visual quality the same as the original image and without inducing artifacts. Interpolation helps to increase or decrease the number of pixels in an image based on local characteristics such as neighboring pixels, edge information, etc.

Table 2
Comparison of iBuck with other existing research studies.

Approach	Image resizing	Encryption/ decryption	PJPEG feature	Underlying methodology	Cloud deployment	Support to diversified remote devices
iBuck	Yes	Yes	Yes	Interpolation method, PFCC for security	Yes	Yes
Liu et al. (2020)	No	Yes	No	MVSSE scheme	Yes	No
Pan et al. (2021)	No	Yes	No	CNN-based hashing	No	No
Nam et al. (2019)	Yes	No	No	Deep neural network	No	No
Danon et al. (2021)	Yes	No	No	Convolutional neural network	No	No
Toderici et al. (2017)	No	No	Yes	Recurrent neural network	No	No
Marwan et al. (2017a)	No	Yes	No	Segmentation and watermarking	Yes	No
Marwan et al. (2017b)	No	Yes	No	Shamir's secret share scheme	Yes	No
Wang and Yuan (2014)	Yes	No	No	Logistic regression	No	No
Hani et al. (2014)	No	No	No	ownCloud	Yes	No
Liu and Sung (2009)	Yes	No	No	Correlation of neighboring DCT	No	No



Fig. 5. Several versions of same images are needed to cover diversified devices such as desktop, laptop, iPad, mobile, etc.

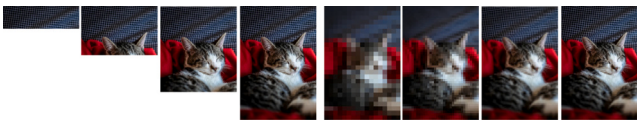


Fig. 6. (a) Baseline download of an image, (b) Progressive download of an image (Pock, 2019).

4.1.1. Choosing the best interpolation method

In our study, we mostly adopt the interpolation methods supported by Python-pillow (Lundh and Clark, 2021). Major image processing-related tasks in the pillow are done using Non-Adaptive interpolation methods. Among them, the most popular methods are Nearest Neighbor, Bilinear, Bicubic, and Antialias. These methods use adjacent 0 to 256 pixels for interpolating which depends on the complexity of the underlying algorithm. Hence, both the computational complexity, processing time, and accuracy increase according to the increase in the number of pixels while doing the task of interpolation. Therefore, there remains a trade-off between computational complexity and accuracy (Roszkowiak et al., 2017). Hence, the smoothness of a resized image depends on how sophisticated the interpolation algorithm is and how it strikes the trade-off.

In our study, we first investigate the performance of resizing images using all of the four types of interpolation methods. Table 5 presents the comparison over qualitative results of various interpolation methods (Roszkowiak et al., 2017). Among them, Bicubic appears to be the most suitable one for resizing, as it gives smoother images in lower time (experimental results are presented in Section 5). The example codebase of the four interpolation methods is given below.

```
import cStringIO
from PIL import Image
img = Image.open(cStringIO.StringIO(<Image file path>))
resize_image = cStringIO.StringIO()
out = img.resize((<new_width>,<new_height>),
PIL.Image.<Interpolation methods>)
out.save(resize_image, 'JPEG')
```

Besides, baseline and progressive are the two primary types of JPEG images. Nowadays, the majority of the websites use baseline JPEGs, where the image appears in a top to bottom manner; i.e the image starts loading with the fully rendered top of the image and then gradually draw the rest according to the data received. On the other hand, progressive JPEGs load the full photo in a pixelated manner, i.e it loads AC and DC particles of the image scan by scan using the scan-script provided in the JPEG library.

4.2. Creating progressive JPEG

Progressive JPEG is generally very fast, smooth, and gives better user experiences. Depending on the file size, progressive JPEGs can be much better optimized resulting in a smaller output size (leading to faster load times).

Modern browsers can display PJPEG images more quickly as the size is lower (in Figs. 6a, and 6b). We add a new middleware in our proxy server for resizing the images to several dimensions and save the resized images both in JPEG and PJPEG format. We use python-pillow for saving the newly resized images as progressive images. An example codebase of our implementation is given below.

```
img = Image.open(<image file path>)
progressive_img = StringIO()
img.save(progressive_img, 'JPEG', quality=80, optimize=True, progressive=True) # Image save to progressive.
```

Nonetheless, we also perform image encryption and decryption in the middleware using PFCC algorithm (Noor et al., 2017; Noor and Al Islam, 2017; Torres et al., 2018).

4.3. Encryption and decryption of images

In the field of cryptography, every modern and classic algorithm has their own advantages and disadvantages. Examples of such modern algorithms are Vigenere Cipher (Kumari et al., 2017), Data Encryption Standard (DES) (Kumari et al., 2017), International Data Encryption Algorithm (IDEA) (Kumari et al., 2017), Blowfish (Kumari et al., 2017), Visual Cryptography (Kumari et al., 2017), RC4 (Kumari et al., 2017), RC5 (Kumari et al., 2017), RC6 (Kumari et al., 2017), Triple Data Encryption Standard (TDES) (Kumari et al., 2017), Advanced Encryption Standard (AES) (Kumari et al., 2017), etc. These types of algorithms are more complex to use when batch parallel processing is needed in cloud computing and images are delivered to mobile cloud users. We present a qualitative comparison over the algorithms (Kumari et al., 2017) in Table 3. Further, we perform experimental evaluation over the algorithms in our considered framework to choose the best one. We will present outcomes of the experimentation later in Section 5.

Table 3
Qualitative comparison over encryption–decryption algorithms as per the study in Kumari et al. (2017).

Encryption schemes	Visual scrambling	Histogram distribution	Correlation coefficient values	Pixel sensitivity	Key space	Key sensitivity	PSNR	Entropy	Time complexity
Vigene're (Kumari et al., 2017)	Least	Spiked	High	Least	Moderate	Low	Least	High	Least
DES (Kumari et al., 2017)	Moderate	Spiked	Moderate	Moderate	Low	High	High	High	High
IDEA (Kumari et al., 2017)	Moderate	Spiked	Moderate	Moderate	Moderate	High	High	High	High
Blowfish (Kumari et al., 2017)	Moderate	Spiked	Moderate	Moderate	Low	High	High	High	Moderate
Visual (Kumari et al., 2017)	High	Uniform	Very low	Least	Moderate	Least	High	High	Least
RC4 (Kumari et al., 2017)	High	Uniform	Very low	Least	Moderate	High	High	High	Least
RC5 (Kumari et al., 2017)	Moderate	Spiked	Moderate	Moderate	Moderate	High	High	High	High
RC6 (Kumari et al., 2017)	Moderate	Spiked	Moderate	Moderate	Moderate	High	High	High	High
TDES (Kumari et al., 2017)	Moderate	Spiked	Moderate	Moderate	Moderate	High	High	High	Maximum
AES (Kumari et al., 2017)	Moderate	Spiked	Moderate	Moderate	Moderate	High	High	High	Low
PFCC (Torres et al., 2018)	Least	Uniform	Moderate	Least	Low	Low	High	High	Least

4.3.1. Choosing best encryption–decryption algorithm

To ensure the security of data in big data or handling large-scale web services for any large-scale computation is a key challenge now. The same challenge applies to images in the cloud. Hence, a research study (Torres et al., 2018) proposes an image encryption service for OpenStack object storage environment to ensure authentication and privacy based on p-Fibonacci Transformation of Cosine Coefficients (PFCC). As this algorithm is more secure and faster for batch processing, we adopt PFCC for image encryption and decryption. In PFCC, Discrete Cosine Transform (DCT) (Agaian et al., 1995) and 2D P-Fibonacci transform (Agaian et al., 1995) are used for its good 'energy compaction' property. Here, most of the signal information tends to be concentrated in a few low-frequency components. DCT's energy compaction performance closely resembles that of KLT, however, it can be implemented via fast algorithms (Torres et al., 2018; Noor and Al Islam, 2017; Noor et al., 2017). After, DCT energy compaction and P-Fibonacci transform are performed.

The forward 2-D DCT-II of an image $f(x, y)$ is given by Noor and Al Islam (2017), Noor et al. (2017):

$$F(u,v) = \frac{1}{4}C(u)C(v)\sum_{x=0}^{7}\sum_{y=0}^{7} f(x,y)\cos\left[\frac{\pi(2x+1)u}{16}\right] \times \cos\left[\frac{\pi(2y+1)v}{16}\right]; \text{ for u = 0,1,...,7 and v = 0,1,...,7}$$

$$\text{where } C(k) = \begin{cases} 1/\sqrt{2} & \text{for } k = 0 \\ 1 & \text{otherwise} \end{cases} \quad (3)$$

Besides, inverse Discrete Cosine Transform IDCT-II is given by Noor and Al Islam (2017), Noor et al. (2017):

$$f(x,y) = \phi(N)\sum_{i=0}^{N-1}\sum_{j=0}^{N-1} F(u,v)\cos\left[\frac{\pi}{N}\left(u+\frac{1}{2}\right)x\right]\cos\left[\frac{\pi}{N}\left(v+\frac{1}{2}\right)y\right]$$

where,

N = No of blocks

$\phi(N)$ = Constant value based on N

$x, y = 0, 1, 2, \ldots, N-1$

4.3.2. PFCC for data encryption

Encryption methods enable the security of data by converting it into a more complex form. It becomes important to reduce the size of the data by preserving the complexity. In our study, we adopt an encryption methodology based on DCT and Fibonacci transform (Torres et al., 2018; Noor and Al Islam, 2017; Noor et al., 2017) for ensuring the security of images. Image encryption methodology contains four processes namely image decomposition, bit-plane shuffling, bitplane encryption, and bit-plane mapping to RGB plane. At first, the DCT coefficients of an image are computed into a matrix using the proxy service of OpenStack Swift for storing the encrypted image in the cloud. Next, it computes the Fibonacci P-code decomposition in 24 bit-planes of the DCT coefficient values as described in paper (Torres et al., 2018; Noor and Al Islam, 2017; Noor et al., 2017).

Finally, using a secure private key, the 24 bit-planes of the image are mapped to encrypted RGB planes by the proxy service. This encryption algorithm illustrates a dual encryption method conducted by the proxy service using P-Fibonacci Transformation of the DCT coefficient and private keys. Authorized users have the security keys for recovering the original image from the encrypted image. The decryption process is similar to the encryption process, however, in a reverse manner.

4.4. Deploying a new middleware in OpenStack Swift

We propose to have a private cloud instead of using public clouds in our secured offline pre-processing-based solution. Here, we propose a new cloud framework by deploying new middleware in OpenStack Swift. Thus, our proposed methodology comprises the following:

- We use PIL's Bicubic algorithm (Lundh and Clark, 2021) for resizing each image into four different sizes in our proposed private cloud. To the best of our knowledge, the best algorithm for resizing is Antialias, however, it is slower than Bicubic. Accordingly, we develop our middleware in such a way that it performs the operations of resizing and encryption–decryption as needed. Algorithm 1 shows our algorithm for resizing an image to the given dimension. Besides, Algorithm 2 shows the algorithm of uploading an image. While uploading an image, we perform encryption using the PFCC algorithm. While downloading an image, we perform the necessary decryption.
- All resized images have a PJPEG version. To do all these things, we add iBuck middleware in several source files of Swift including setup.cfg, proxy-server.conf, and middleware package. Algorithm 3 shows the algorithm of our implemented iBuck middleware.

In our implementation of iBuck middleware, we develop several codebases for enabling diversified functionalities of the middleware. To do so, we include an entry point for the middleware, add the middleware to the proxy server. We present these two aspects of development next.

4.4.1. Adding iBuck entry-point in setup.cfg file

setup.cfg is the configuration file of OpenStack Swift. All the middleware of Openstack Swift is called using a filter factory pattern in Paste package (Arnold, 2015). Hence, each middleware should be included

in the entry-point section in the setup.cfg file. We present the code for setup.cfg file pertinent to our implementation below:

```
[metadata]
name = swift
...
[pbr]
skip_authors = True
...
[files]
packages = swift
scripts = bin/swift-account-audit
...
[entry_points]
paste.filter_factory = healthcheck =
swift.common.middleware.healthcheck:filter_factory
...
ibuck = swift.common.middleware.ibuck:filter_factory
% Our implementation
...
```

Here, "[ ]" denotes several sections for setup.cfg file, and "…" refers existence of more codes in the file that we omit here.

4.4.2. Adding iBuck middleware in Proxy-Server.conf file

Proxy-Server.conf is the proxy server configuration file of OpenStack Swift. The majority of the middleware implementations are operated through the proxy server. Our proposed iBuck middleware also operates through the proxy server. Hence, we need to include it in Proxy-Server.conf file. We present the code of Proxy-Server.conf file pertinent to our implementation below:

```
[DEFAULT]
# bind_ip = 0.0.0.0
bind_port = 8080
...
[pipeline:main]
pipeline = catch_errors gatekeeper healthcheck
proxy-logging cache ··· ibuck ··· proxy-logging
proxy-server % our implementation
[app:proxy-server]
use = egg:swift#proxy
[filter:catch_errors]
use = egg:swift#catch_errors
[filter:healthcheck]
use = egg:swift#healthcheck
[filter:tempauth]
use = egg:swift#tempauth
[filter:cache]
use = egg:swift#memcache
[filter:ibuck] % our implementation
use = egg:swift#ibuck % our implementation
...
```

Here, in the [pipeline:main] section, all the middleware implementations are included and separated by spaces. Order of calling middleware implementations is maintained through this pipeline. In the pipeline, catch_errors, gatekeeper, health check, proxy-logging, cache, etc. are the middleware implementations of Openstack Swift. Here, iBuck middleware is our developed middleware. Figs. 7, 8, and 9 present the flow diagram of installation, execution steps, and overall architecture of iBuck middleware.

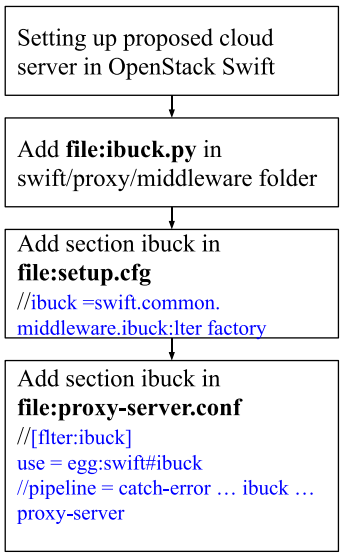


Fig. 7. Flow diagram of implementation steps.

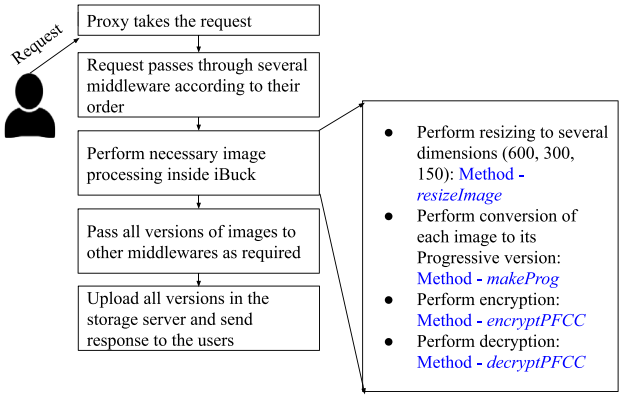


Fig. 8. Flow diagram of execution steps of iBuck middleware.

4.5. Request analysis of proposed architecture

After deploying the new middleware in our proposed cloud server, we test our server using various HTTP requests. Here, we perform account authentication, account verification, account creation, container creation, container listing, object upload, image upload, object download, image download, etc. We test the proposed image cloud framework in two setup test-beds by operating with a bulk amount of images. We present operational outcomes of image resizing and the time needed for uploading the images in the next section.

5. Experimental evaluation and comparative analysis

We evaluate the performance of our proposed methodology through a real implementation. Before presenting the evaluation results, we first briefly elaborate on our experimental settings.

5.1. Experimental setup

To compare the efficiency of our proposed private cloud model with these implementations, we divide our experimental setup into two different parts. One is for Quality of Experience (QoE) based evaluation setup, and another is for physical test-bed server setup for system-level evaluation. In the below sections, we present these two types of experimental setups.

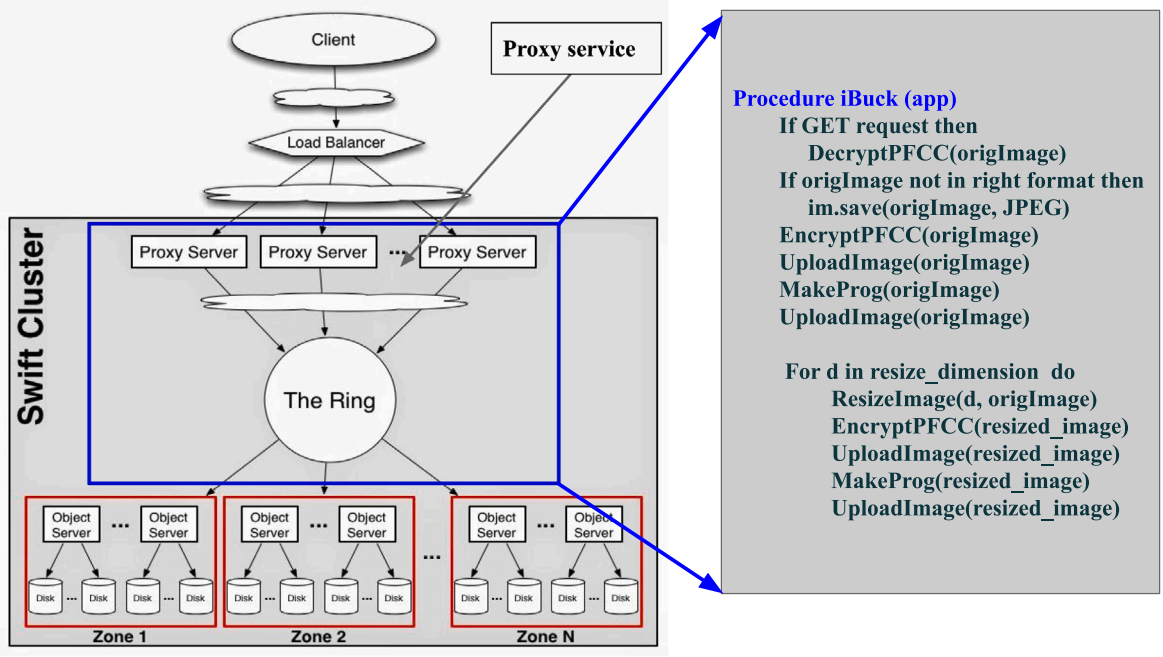
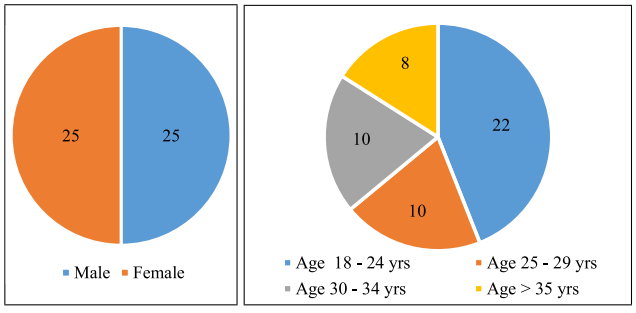


Fig. 9. Overall architecture of iBuck middleware.

Table 4
Information of files used for image quality testing.

Quality test file	Size (kb)	Type	Resolution
Image 1	360	JPG	1280 × 720
Image 2	540	PNG	1280 × 720
Image 3	460	JPG	1280 × 720
Image 4	304	JPG	1400 × 1050
Image 5	105	PNG	480 × 270



(a) Gender (b) Age

Fig. 10. Demography of our subjects/observers.

5.1.1. QoE evaluation setup

Table 4 shows information about files used for quality measurement. We evaluate the quality of the images after applying the different algorithms in two different ways namely subjective evaluation (Wang et al., 2017) and objective evaluation (Wang et al., 2017).

- **Subjective evaluation:** Here, an image is presented to a subject (person) to be watched by him or her to evaluate the quality of resized images. In our study, we invite 50 observers to observe resized images and collect their evaluations. We use average scores to evaluate the subjective image quality. Fig. 10 presents the demography of the subjects, which presents variations in genders and ages of the observers.
- **Objective evaluation:** For objective evaluation, we evaluate our middleware algorithm in two different ways as follows:
  - Using two QoE metrics, and
  - Using time and size needed for resizing PJPEG images after conversion using various interpolation methods.

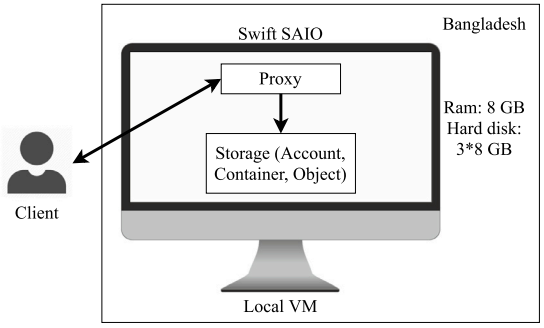
  We compare the time required for converting an image using various interpolation methods. Here, we take 50 images of JPG and PNG types, and resize them using Python Pillow interpolation methods. Besides, we take some specific images — JPEG type images (approximate dimension: 300 × 300; size: 30 kb), and PNG type images (approximate dimension: 1400 × 1050; size: 1540 kb). We run the algorithm 10 000 times over these images and take averages of the experimental results.

5.1.2. Test-bed server setup

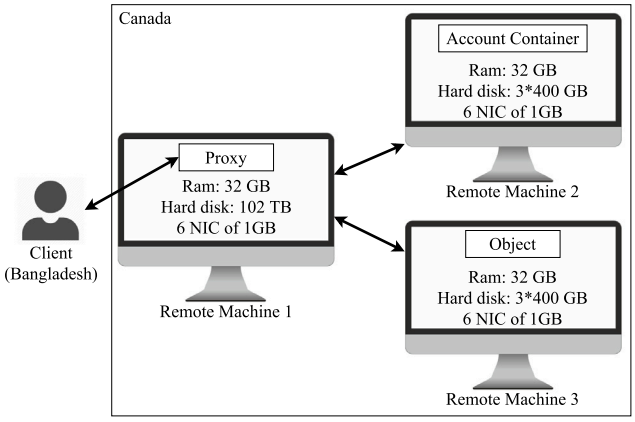
Furthermore, we evaluate the time needed for storing image files in our proposed image cloud storage server. Table 4 shows information about the image files used for evaluating the uploading time. We perform the uploading by conducting our experiments in two different setups.

In our experiment, we set up native OpenStack Swift Storage as - (1) similar to what the conventional public clouds do without any change in the original code, and (2) following our proposed architecture through deploying the new middleware as a private cloud. Moreover, we compare the conventional public cloud setup (without using middleware) and our proposed cloud architecture (with using middleware) in a similar deployment environment to make sure impacts of other factors such as network bandwidth, CDN in operation, etc., remain the same in both cases. To do so, we deploy both of them in the local server and in the remote server having the same Internet bandwidth without enabling any CDN.

We deploy one setup in a local machine in Bangladesh, where servers are setup in a local virtual machine. Another setup is in a remote data center in Canada. For each setup, we install three different servers — one for proxy, one for account-container, and the last one for object server (Arnold, 2015). Fig. 11 depicts the server setup for our proposed framework. The memory and disk configurations of testing servers are as follows:



(a) Local server setup in virtual machine



(b) Remote server setup

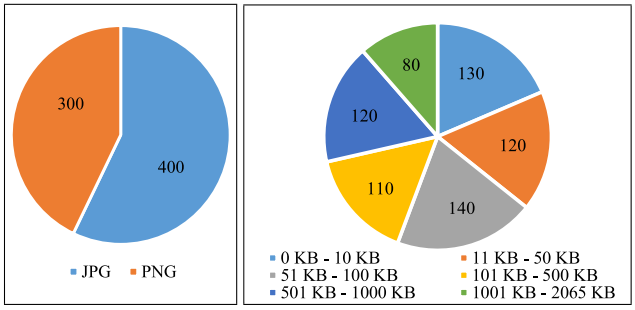
Fig. 11. Server setup for our proposed framework.

- **Local server:** Proxy, account-container, and object servers are installed in a local virtual machine having CPU 16, model Intel(R) Xeon(R) CPU E5620 @2.40GH Processor 6174, eight GB memory, and three disks each of eight GB. The machine is network-connected through a bridge connection having a speed of 0.5 Mbps.
- **Remote server:** The remote server consists of one proxy having 32 GB memory and one 1.2 Tb disk, one account-container having 32 GB memory and three disks each of 400 GB, and one object having 32 GB memory and three disks each of 400 GB. Each server has six 1 GB network interface cards. Each server having CPU 16, model Intel(R) Xeon(R) CPU E5620 @2.40GH Processor 6174.

The average uploading and downloading network speed were around 1.95 Mbps and 2.1 Mbps respectively when we conduct our experiments. We upload 700 images for finding out the results of encryption–decryption time and uploading–downloading performance of our proposed iBuck middleware. We present the demography of our image data set in Fig. 12.

5.2. Experimental results

After completing the physical server setup, we perform the experiments in our test-bed server. At first, we use our local machine for finding out the best interpolation method. As previously discussed, we use both subjective and objective evaluation for finding the best interpolation method. Subsequently, we deploy our new middleware using the best interpolation method. Besides, we present a comparison on time for uploading an image with or without using our proposed middleware. All key steps are given in the below sections — We present all steps of our experimentation and their results in the following subsubsections.



(a) Image type (b) Image size

Fig. 12. Demography of our data set.

Table 5
Subjective evaluation of interpolation methods for resizing an image.

Interpolation method	Subjective feelings	Image contour	Overall evaluation	Time (ms)
Nearest neighbor	Obvious mosaic phenomenon	Not clear	Worst	20
Antialias	Relatively clear and sharp	Clear	Best	126
Bilinear	Blur and not sharp	Not clear and serrate phenomenon	Poor	24
Bicubic	Fuzzy and sharper	Serrate phenomenon has improved Edge	Good	30

Table 6
MOS value of five images obtained from 50 observers.

Image	Nearest neighbor	Bilinear	Bicubic	Antialias
Image1	1.9	3.9	4.3	4.3
Image2	2.1	3.1	4.2	4.45
Image3	1.88	2.9	4.7	4.8
Image4	2.2	3.2	4.9	5
Image5	2.8	3.25	5.0	5
Average	2.12	3.27	4.62	4.71

5.2.1. Results of resizing through subjective evaluation

We use Mean Opinion Score (MOS) metric for subjective based evaluation. Here, we convert original images to several interpolated images using Bicubic, Nearest Neighbor, Bilinear, and Antialias interpolation methods.

Fig. 13 presents interpolated versions of a sample image. We invite 50 observers to watch such interpolated images and collect their opinions on the interpolated images. We collect information related to subjective feelings, image contour, and overall evaluation from the observers. Table 5 shows a comparison over the methods in terms of these metrics along with the time required for resizing an image. We present here opinions obtains from ten images. Besides, Table 6 presents MOS values obtained from 50 observers for five interpolated images. We already present that the MOS value ranges from 0 to 5, where 0 refers to the worst, and 5 refers to the best user experience.

Table 6 shows that observers give the lowest value for the images that are converted using the Nearest Neighbor. For Bilinear conversion, image quality appears to be average. Both Bicubic and Antialias type images exhibit to attain good ranking according to the observers.

5.2.2. Results of resizing through objective evaluation

For objective evaluation, we use two different methods for finding out the best interpolation method. We present the outcomes of the evaluation below.



(a) Original image (b) Bicubic image (c) Nearest image (d) Bilinear image (e) Antialias image

Fig. 13. Results of different interpolation methods after applying on Image-1. (a) Original Image. Interpolated images by (b) Bicubic, (c) Nearest Neighbor, (d) Bilinear, and (e) Antialais methods.

Table 7
MSE values of the interpolated images calculated using VQMT (Vatolin et al., 2021).

Image	Nearest neighbor	Bilinear	Bicubic	Antialias
Image1	410.6	188.9	187.8	186.7472
Image2	115.9	42.2	39.3	39.4
Image3	98.6	30.8	26.3	27.3
Image4	388.3	210.8	208.8	210.3
Image5	195.9	67.8	63.6	62.3

Table 8
SSIM values of the interpolated images calculated using VQMT (Vatolin et al., 2021).

Image	Nearest neighbor	Bilinear	Bicubic	Antialias
Image1	0.7849	0.8238	0.8634	0.8632
Image2	0.8087	0.8137	0.8775	0.8702
Image3	0.8324	0.9129	0.9145	0.9126
Image4	0.7104	0.7332	0.7952	0.7654
Image5	0.8277	0.9112	0.9116	0.9113

- **Using QoE metrics:** We use Mean Squared Error (MSE) and Structural Similarity Index (SSIM) for calculating image quality after resizing. For calculating MSE and SSIM, we use VQMT software (Vatolin et al., 2021). Tables 7 and 8 present the MSE and SSIM values for four interpolated images. Table 7 presents that both Bicubic and Antialias images exhibit lower MSE values compared to that of Nearest Neighbor and Bilinear images. Here, the Nearest Neighbor images exhibit the highest MSE values.
  Besides, Table 8 presents that both Bicubic and Antialias images exhibit higher SSIM values compared to that of Nearest Neighbor and Bilinear images. Here, Bicubic images exhibit the highest SSIM values.
- **Using size and time needed for resizing PJPEG images:** After choosing the best interpolation method, we calculate the size and time needed for different versions of images. We perform this calculation for both PNG and JPG types of images.
  As we plan to convert all images to JPEG and PJPEG, we resize the original image of both the types (PNG or JPG) to three different versions as follows:

  1. 600 to aspect ratio for covering middle size devices,
  2. 300 to aspect ratio for covering lower size devices, and
  3. 150 to aspect ratio for covering thumb images.

Figs. 14a, and 14b show the size and time comparison of PNG type images that are resized by our middleware to three different dimensions and subsequently saved in JPEG and PJPEG. Here, we use some PNG images (average 1540 kb) which are reduced to 639.47 kb after converting to PJPEG.

Besides, Figs. 15a, and 15b show the size and time comparison of JPG-type images that are resized by our middleware to three different dimensions and subsequently saved in JPEG and PJPEG. Here, we use some JPEG images (average 300.22 kb) which are reduced to 204.99 kb after converting to PJPEG.

Figs. 14a, and 15a demonstrate that using iBuck middleware, can marginally increase the size of PNG images in few cases, however,

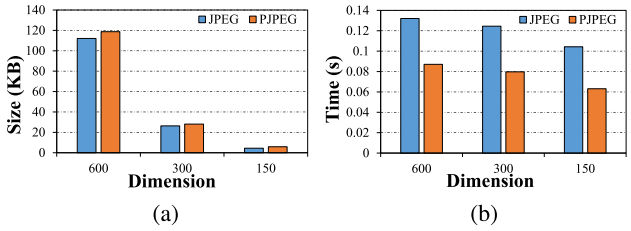


Fig. 14. (a) Present the PNG type images size after converting to different versions of both JPEG and PJPEG, (b) The Time needed for converting PNG type images to different versions of both JPEG and PJPEG.

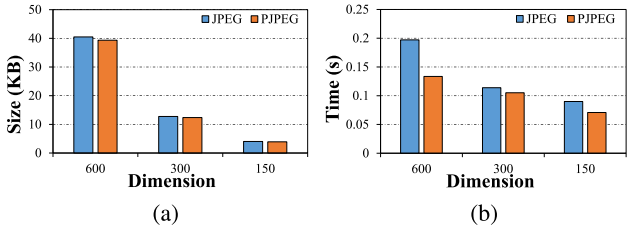


Fig. 15. (a) Present the JPG type images size after converting to different versions of both JPEG and PJPEG, (b) The Time needed for converting JPG type images to different versions of both JPEG and PJPEG.

it always decreases the size of JPG images. Here, the sizes pertinent to our iBuck middleware are the sizes of PJPEG images, which are always decreased in the case of JPG images. More importantly, using the middleware significantly reduces the time required for resizing the images. The improvement in time requirement can be up to 18% for PNG and up to 25% for JPEG images. Size can be optimized up to 60% for PNG and up to 38% for JPEG images when we save images as PJPEG.

5.3. Results of image encryption–decryption

The classic algorithms have some drawbacks of consuming more time and of not being applicable for parallel batch processing inside a cloud file environment due to higher time consumption (given in Table 3). On the contrary, the PFCC algorithm (Torres et al., 2018; Noor and Al Islam, 2017; Noor et al., 2017) has been proposed for parallel image encryption–decryption in batch mode.

To investigate the suitability of the encryption–decryption scheme in our proposed cloud model, we perform experimentation through encrypting and decrypting 700 images in batch mode in our test-bed setup. We present the outcomes of the experiment in Fig. 16, which suggests that the PFCC algorithm outperforms all other alternatives under consideration in terms of both encryption time and decryption time. Moreover, Fig. 17 presents the data size after the encryption. Considering all these aspects, we adopt the PFCC algorithm in our proposed cloud model.

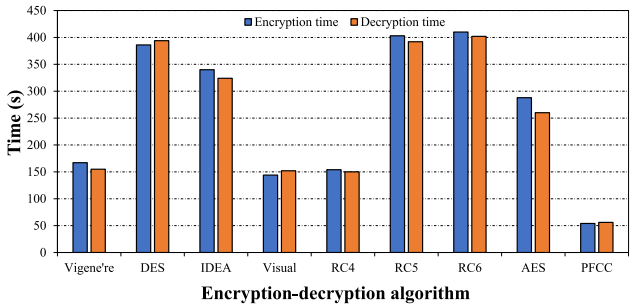


Fig. 16. Time comparison of different encryption–decryption algorithms for 700 images.

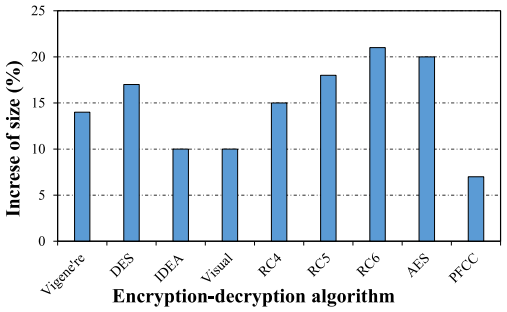


Fig. 17. % of Size increased after encrypting 700 images (total 1156 MB before image encryption) through applying different encryption algorithms.

5.4. Results of overall middleware implementation

Finally, we evaluate the time requires for storing an image using iBuck middleware comprising all the tasks of resizing and encryption–decryption. We do so through conducting experiments over two different setups given in Section 5.1.2. We store one version of an image for evaluating performance *without* using the implemented iBuck middleware. However, we store eight different versions of an image in the server to analyze the performance of using the iBuck middleware in our cloud server. The versions of an image are as follows:

- Original image (main)
- Progressive version of the original/main image (main progressive)
- 600 version of the original/main image
- Progressive 600 version of the original/main image
- 300 version of the original/main image
- Progressive 300 version of the original/main image
- 150 version of the original/main image
- Progressive 150 version of the original/main image

At first, we upload only the main image without enabling our iBuck middleware. Later, we convert these versions from the original image then upload each version of the image to our test-bed server in parallel without enabling our iBuck middleware and measure how much time is needed for conducting the uploading task. Besides, we enable our iBuck middleware and upload the images, where all versions of the images get uploaded in the cloud through the middleware. We upload a total of seven hundred images.

We find the average time needed for uploading images to be 0.3280 s and 0.6699 s for the local and remote servers respectively when we upload only the main image without enabling the iBuck middleware. The time needed for uploading the eight versions of an image is 0.3939 s and 4.9183 s for the local and remote servers respectively without using the iBuck middleware. While using the iBuck middleware, these time values become 0.3343 s and 0.6789 s. Fig. 18 shows a comparison

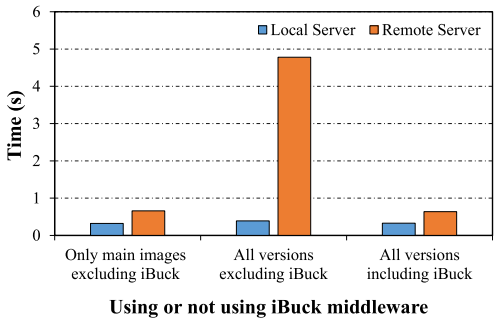


Fig. 18. Time required for storing images with or without using iBuck middleware for our own image data set.

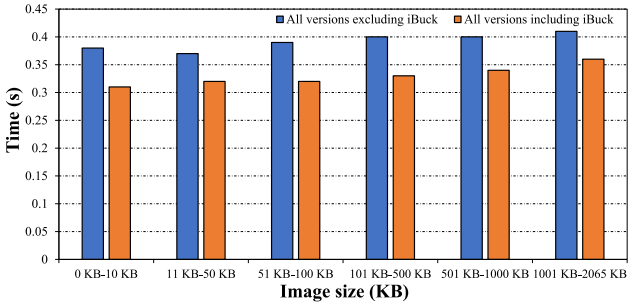


Fig. 19. Time required in local server for storing images with or without using iBuck middleware for different sizes of images.

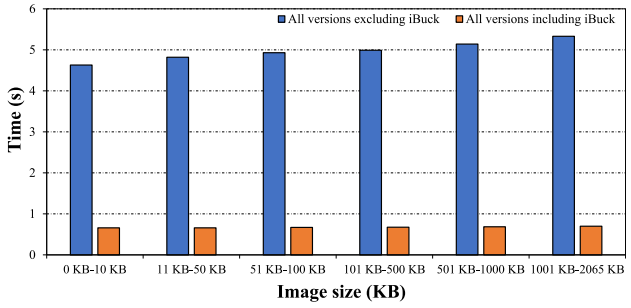


Fig. 20. Time required in remote server for storing images with or without using iBuck middleware for different sizes of images.

of the time for storing images. These results indicate that the average storing time significantly decreases while using iBuck middleware as we store the images after necessary conversions through operations integrated within the middleware. Here, we can achieve up to 17% and up to 80% improvement in storing time for local server and remote server respectively.

In addition, Figs. 19 and 20 present the time required for storing an image with or without using iBuck middleware for different sizes of images in local and remote servers respectively. These results demonstrate that our performance improvement in storing time with iBuck middleware is invariant of different sizes of images.

Besides, we also perform downloading images from a remote server while using our iBuck middleware and while using a public cloud namely cloudinary (Lahan, 2021). We setup an experimental public cloud in the same server that performs on-the-fly image processing. We download several versions of 400 images. We present the average downloading time in Fig. 21, which demonstrates that we can achieve up to 33%, 58%, 65%, 66%, 67%, 75%, and 78% improvement in downloading time for the progressive main image, 600 version of the image, progressive 600 version of the image, 300 version of the image,

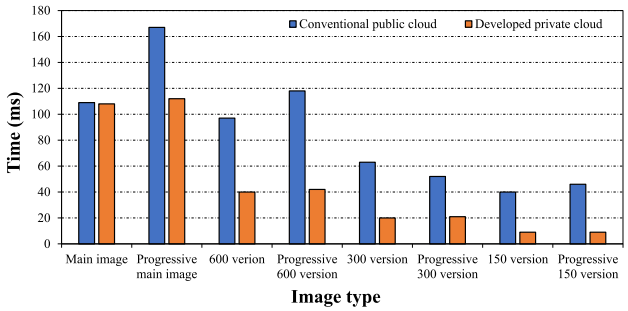


Fig. 21. Time required for downloading image with or without using iBuck middleware.

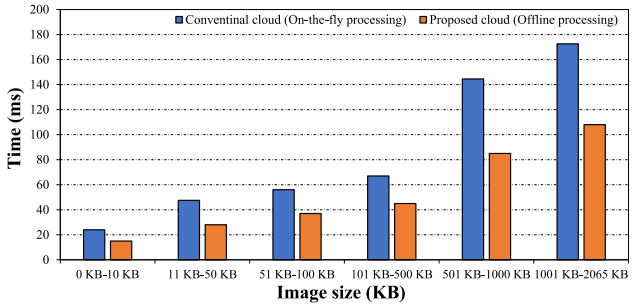


Fig. 22. Time required in remote server for downloading images with and without using iBuck middleware for different size of images taken from our own image data set.

progressive 300 version of the image, 150 version of the image, and progressive 150 version of the image respectively using the iBuck middleware for the remote server. Additionally, Fig. 22 presents the time required for downloading with and without iBuck middleware for different sizes of images. The results demonstrate that our performance improvement achieved in downloading time with iBuck middleware is independent of the sizes of the images.

5.5. Results of overall middleware implementation for standard data sets

In addition to experimenting with our own image data set, we also take three standard data sets for evaluating the performance of the overall middleware implementation. We present demographies of the standard data sets and evaluation results below.

5.5.1. Results of standard data set-1

We take our first standard data set having a size of a total of 40 MB (Tyleček and Šára, 2013). There are 378 JPG and 378 PNG types of images in the data set. Here, the size of JPG-type images ranges from 25 kB to 174 kB, where their minimum dimension is 350 × 400 and maximum dimension is 753 × 1024. For PNG type of images, size ranges from 3 kB to 16 kB, where minimum dimension is 350 × 400 and maximum dimension is 753 × 1024 (Tyleček and Šára, 2013). We upload these images to our proposed server with and without using iBuck middleware. Fig. 23a presents the time required for storing these images with and without using iBuck middleware. Besides, we evaluate the downloading time of on-the-fly processing (from the conventional cloud) and offline processing (from our proposed cloud). Fig. 24a presents the time required for downloading images with and without using iBuck middleware for the standard data set. These experimental results on standard data set-1 demonstrate similar performance improvement as we find for our image dataset.

Table 9
Type-wise demographic information of different standard data sets.

Datasets	JPG	PNG
Dataset-1	378	378
Dataset-2	240	240
Dataset-3	25 279	0

Table 10
Image sizes of different standard data sets.

Datasets	0 kB–10 kB	10 kB–100 kB	100 kB–1000 kB
Dataset-1	435	546	153
Dataset-2	240	0	0
Dataset-3	5	25 258	16

5.5.2. Results of standard data set-2

We take our second standard data set (Korč and Förstner, 2009) having a size of total 78 MB and a demography as presented in Tables 9 and 10. There are 240 JPG and 240 PNG types of images in the data set. Here, the size of JPG types of images ranges from 238 kB to 413 kB, where their minimum dimension is 450 × 500 and maximum dimension is 512 × 786. For PNG type of images, size range are between 3 kB to 9 kB where minimum dimension is 450 and maximum dimension is 512 × 786 (Korč and Förstner, 2009). We upload these images to our proposed server with and without using iBuck middleware. Fig. 23b presents the time required for storing these images with and without using iBuck middleware. Besides, we evaluate the downloading time of on-the-fly processing (from the conventional cloud) and offline processing (from our proposed cloud). Fig. 24b presents the time required for downloading images with or without using iBuck middleware for the standard data set-2. These experimental results on standard data set-2 demonstrate similar performance improvement as we find in our earlier cases.

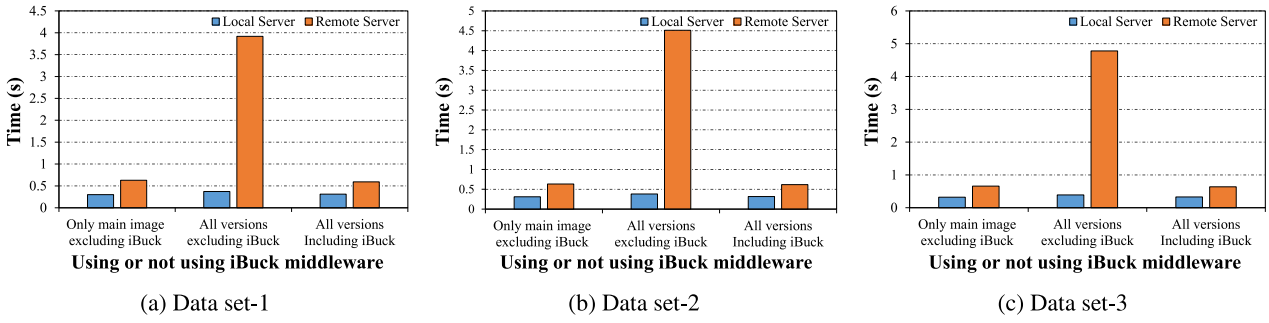
5.5.3. Results of standard data set-3

We take our third standard data set having a size of a total of 825 MB and demography as presented in Tables 9 and 10. There are 25 279 JPG types of images in this data set. Here, image size ranges from 8.33 kB to 116 kB, where the minimum dimension is 600 × 338 and the maximum dimension is 600 × 600. We upload these images to our proposed server with and without using iBuck middleware. Fig. 23c presents the time required for storing these images with and without using iBuck middleware. Besides, we evaluate the downloading time of on-the-fly processing (from the conventional cloud) and offline processing (from our proposed cloud). Fig. 24c presents the time required for downloading the image with and without using iBuck middleware for standard data set-3. These experimental results on standard data set-3 demonstrate similar performance improvement as we have already found in our earlier cases.

6. Discussion and limitations

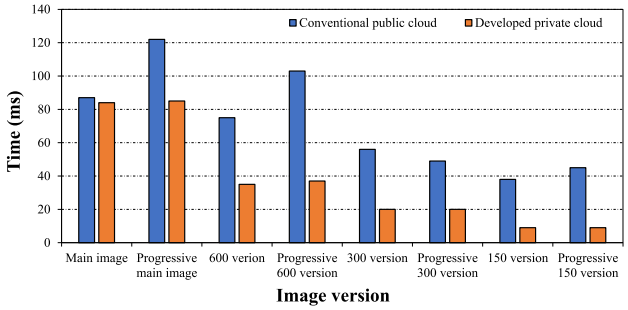
When we upload images without enabling iBuck middleware, only the main images get uploaded to the server. As three copies of each image generally get stored in the cloud storage to ensure availability, the occupied total disk size on the cloud should be three times the total size of the uploaded images without iBuck middleware. On the contrary, when using iBuck middleware, multiple versions get stored for each of the uploaded images. Thus, using iBuck middleware should present a cost of increased cloud storage occupancy. This refers to the expense against which our proposed framework offers all its benefits.

Now, to quantify the expense, we attempt to measure disk size occupancy in our experimentation — both while using iBuck middleware and not using iBuck middleware. As it is extremely difficult (or nearly impossible) for us to measure the disk occupancy in the remote setup

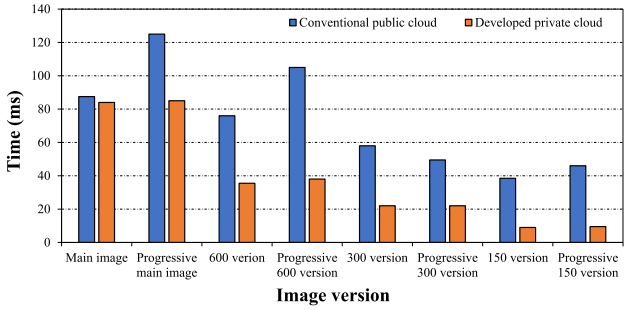


(a) Data set-1 (b) Data set-2 (c) Data set-3

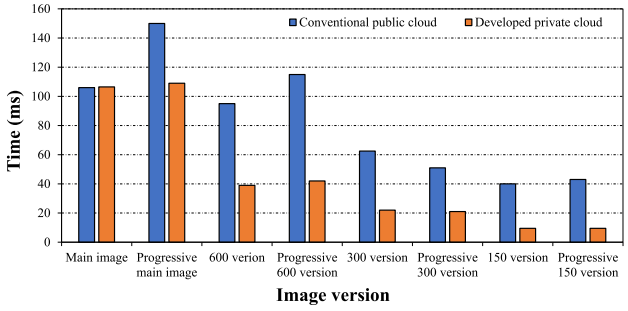
Fig. 23. Time required for storing images with or without using iBuck middleware for standard data sets.



(a) Data set-1



(b) Data set-2



(c) Data set-3

Fig. 24. Time required for downloading images with or without using iBuck middleware for standard data sets.

for having the disks distributed over several different locations, we take the measurements in our local setup.

In our experimentation, the total size of uploaded images is 1156 MB. Hence, the total cloud storage needed in this case is around 3468 MB (1156 MB ×3) for storing the three replicas without iBuck middleware. On the other hand, with iBuck middleware, we measure the size of whole disk occupancy after uploading the images and we find it to be 4521.6 MB (1507.2 MB ×3). Consequently, we find an additional 1053.6 MB (30% more) cloud disk occupancy when using iBuck middleware. This refers to the trade-off that our proposed framework presents between enhanced efficiency, escalated security, and extra need for storage.

Moreover, our proposed architecture is tested against several HTTP requests. Here, our framework considers and examines the notions of correctness and completeness of the requests during its operation. To elaborate, in our framework, a client will send a single request and multiple internal requests will be generated in the processing server of our proposed cloud server. Afterward, a corresponding response will be given to the client. In the process of doing so, our framework is capable of managing data loss. This happens as, after completing all the internal requests, we give a positive response to the client. If any kind of problem arises due to network disruption or disconnection, the request given from the client will be timed out. Then, the client will send the request again following the conventional networking protocol stack and operational paradigm. Similar situations will happen for transitions from the proxy server to the storage (account, container, and object) server. Hence, our architecture is fully capable of managing data loss.

Note that we could not manage to test our server using various HTTPS requests due to certification issues. In the future, our plan is to perform the test cases using several HTTPS requests. We need to explore the overall scenarios using HTTPS requests for better results.

Additionally, in our proposed architecture, there could be two potential attacks by a malicious user in two different network transactions - (1) from the client requests to the proxy server, and (2) from the proxy server to the storage server. For the first case, when we deploy it in a production server, we should use HTTPS to ensure the security of transactions from clients to proxy servers. Besides, clients can send encrypted requests. Accordingly, for the second case, we will need to deploy the storage server in a private network and encrypt the images using PFCC to make the architecture secure. Further, for many other types of attacks such as the DDoS attack, OpenStack Swift is self capable of handling these (for example, OpenStack Swift is self-resilient to DDoS attacks).

7. Future research avenues

In this study, we present a new pre-processing-based cloud model and the key features of the proposed and implemented model. There remain some avenues to extend this study. At first, we plan to implement an efficient, customized, and lightweight encryption–decryption algorithm in place of our currently used PFCC algorithm. Then, our plan includes proposing a better combination of interpolation methods (combining Bicubic and Antialias) where the quality of the image will be higher and processing time will be lower.

Besides, we also plan to develop a new framework for PJPEG that will retrieve the first scan of the image in a smaller amount of time for a better user experience in the future. In our study, we use already-existing Python's implementation of the PIL library. Besides, we use OpenStack Swift, which is also written in Python. In the future, in places of these implementations, we want to implement a new interpolation method for image resizing where image quality will

improve more. Our plan is also to use some techniques for smoothing and enhancing the image quality with lower resolution. In addition, we will include an analysis of experimentation with existing retrieval image databases over our proposed framework. Finally, in addition to the benchmark image databases used in the experimentation of this study, we want to experiment with more benchmark retrieval image databases (Image Retrieval, 2021) in the future.

8. Conclusion

Storing images in the cloud through ensuring their efficient management and security has become crucial in recent times, as many service providers are now delivering their services through storing numerous images over the cloud. Service recipients of such image-based services, on the other hand, are now accessing the services through different client-end devices requiring different versions of the images. To comply with the requirement of delivering the different versions, it is conventionally practiced to process the required versions on the fly as per users' requests.

On the contrary to the conventional practice that adopts a cloud framework performing on-demand image version management, in this study, we propose a new cloud framework enabling pre-processing of the image versions. Apart from enabling the pre-processing, we also investigate the appropriate resizing method and encryption–decryption algorithm for our proposed framework.

While conducting our study, we setup real cloud environments both in local and remote setups. In our real cloud setup, which subsumes a private cloud to alleviate the limitations of the conventional public clouds, we perform rigorous experimentation to find out the best possible image resizing method. Our experimentation reveals that the Bicubic method (Roszkowiak et al., 2017) appears to be the best possible method for our proposed framework. We confirm the suitability of the Bicubic method among the existing alternatives through both subjective and objective evaluations. Besides, we perform experimentation to find out an appropriate encryption–decryption algorithm for our proposed cloud framework. In this regard, PFCC algorithm (Noor et al., 2017; Noor and Al Islam, 2017; Torres et al., 2018) appears to be the best alternative as per our experimental results.

Later, we conduct rigorous experiments on uploading and downloading images both using and without using our proposed framework. In our experimentation, to enable using our proposed framework, we implement a new middleware that we name as iBuck. Besides, we utilize both custom and standard data sets in our experimentation. Our experimental results demonstrate that we can achieve substantial performance improvement using our proposed framework compared to its existing alternatives.

We envision some avenues to extend this study further. For example, we plan to devise a new encryption–decryption algorithm that should be more efficient and lightweight compared to our adopted PFCC algorithm in our proposed framework. We also plan to come up with a better combination of interpolation methods to ameliorate the quality of the image further with less processing time. We also want to explore a new framework for achieving a faster first scan of the images in the future.

CRediT authorship contribution statement

Jannatun Noor: Conceptualization, Methodology, Software, Data curation, Validation. **Saiful Islam Salim:** Writing – original draft, Literature survey, Writing – review & editing. **A.B.M. Alim Al Islam:** Supervision, Writing – review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgment

This work has been supported by the ICT Division, Government of the People's Republic of Bangladesh.

References

Agaian, S., Astola, J., Egiazarian, K., Kuosmanen, P., 1995. Decompositional methods for stack filtering using fibonacci P-codes. Signal Process. 41 (1), 101–110. http://dx.doi.org/10.1016/0165-1684(94)00093-F.

Arnold, J., 2015. OpenStack Swift. O'Reilly, SwiftStack Inc., 978-1-491-90082-6.

Aroussi, S., Bouabana-Tebibel, T., Mellouk, A., 2012. Empirical qoe/qos correlation model based on multiple parameters for vod flows. In: 2012 IEEE Global Communications Conference (GLOBECOM). pp. 1963–1968. http://dx.doi.org/10.1109/GLOCOM.2012.6503403.

Cayirci, E., Rong, C., Huiskamp, W., Verkoelen, C., 2009. Snow leopard cloud: A multi-national education training and experimentation cloud and its security challenges. In: Proceedings of the 1st International Conference on Cloud Computing. In: CloudCom '09, Springer-Verlag, Berlin, Heidelberg, pp. 57–68. http://dx.doi.org/10.1007/978-3-642-10665-1_6.

Chang, V., 2015. Towards a big data system disaster recovery in a private cloud. Ad Hoc Networks 35, 65–82. http://dx.doi.org/10.1016/j.adhoc.2015.07.012, Special Issue on Big Data Inspired Data Sensing, Processing and Networking Technologies.

Chidambaram, N., Raj, P., Thenmozhi, K., Amirtharajan, R., 2020. Advanced framework for highly secure and cloud-based storage of colour images. IET Image Process. 14, http://dx.doi.org/10.1049/iet-ipr.2018.5654, 3143–3153(10).

Danon, D., Arar, M., Cohen-Or, D., Shamir, A., 2021. Image resizing by reconstruction from deep features. Comput. Vis. Media http://dx.doi.org/10.1007/s41095-021-0216-x.

Devaraj, A.F.S., Murugaboopathi, G., Elhoseny, M., Shankar, K., Min, K., Moon, H., Joshi, G.P., 2020. An efficient framework for secure image archival and retrieval system using multiple secret share creation scheme. IEEE Access 8, 144310–144320. http://dx.doi.org/10.1109/ACCESS.2020.3014346.

Díaz, C., Pérez, P., Cabrera, J., Ruiz, J.J., García, N., 2020. XLR (pixel loss rate): A lightweight indicator to measure video QoE in IP networks. IEEE Trans. Netw. Serv. Manag. 17 (2), 1096–1109. http://dx.doi.org/10.1109/TNSM.2020.2980752.

Doelitzscher, F., Sulistio, A., Reich, C., Kuijs, H., Wolf, D., 2011. Private cloud for collaboration and E-learning services: from IaaS to SaaS. Computing 91 (1), 23–42. http://dx.doi.org/10.1007/s00607-010-0106-z.

Dong, X., Zhang, W., Shah, M., Wang, B., Yu, N., 2020. Watermarking-based secure plaintext image protocols for storage, show, deletion and retrieval in the cloud. IEEE Trans. Serv. Comput. 1. http://dx.doi.org/10.1109/TSC.2020.3008957.

Dong, B., Zheng, Q., Qiao, M., Shu, J., Yang, J., 2009. Bluesky cloud framework: An E-learning framework embracing cloud computing. In: Cloud Computing. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 577–582. http://dx.doi.org/10.1007/978-3-642-10665-1_55.

Dong, B., Zheng, Q., Yang, J., Li, H., Qiao, M., 2009. An E-learning ecosystem based on cloud computing infrastructure. In: 2009 Ninth IEEE International Conference on Advanced Learning Technologies. pp. 125–127. http://dx.doi.org/10.1109/ICALT.2009.21.

Futral, W., Greene, J., 2013. Intel®Trusted Execution Technology for Server Platforms: A Guide to more Secure Datacenters. Apress, http://dx.doi.org/10.1007/978-1-4302-6149-0.

Gruschka, N., Iacono, L.L., 2009. Vulnerable cloud: SOAP message security validation revisited. In: 2009 IEEE International Conference on Web Services. pp. 625–631. http://dx.doi.org/10.1109/ICWS.2009.70.

Hani, A.F.M., Paputungan, I.V., Hassan, M.F., Asirvadam, V.S., Daharus, M., 2014. Development of private cloud storage for medical image research data. In: 2014 International Conference on Computer and Information Sciences (ICCOINS). pp. 1–6. http://dx.doi.org/10.1109/ICCOINS.2014.6868433.

Harrison, C., Dey, A.K., Hudson, S.E., 2010. Evaluation of progressive image loading schemes. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems. In: CHI '10, Association for Computing Machinery, New York, NY, USA, pp. 1549–1552. http://dx.doi.org/10.1145/1753326.1753557.

Hassan, A., Liu, F., Wang, F., Wang, Y., 2021. Secure content based image retrieval for mobile users with deep neural networks in the cloud. J. Syst. Archit. 116, 102043. http://dx.doi.org/10.1016/j.sysarc.2021.102043.

Huang, Z., Zhang, M., Zhang, Y., 2019. Toward efficient encrypted image retrieval in cloud environment. IEEE Access 7, 174541–174550. http://dx.doi.org/10.1109/ACCESS.2019.2957497.

Hunt, T., Jia, Z., Miller, V., Szekely, A., Hu, Y., Rossbach, C.J., Witchel, E., 2020. Telekine: Secure computing with cloud GPUs. In: 17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20). USENIX Association, Santa Clara, CA, pp. 817–833.

2021. Image retrieval. [Online]. Available: https://paperswithcode.com/task/image-retrieval. Accessed: May 19, 2021.

Jia, W., Zhu, H., Cao, Z., Wei, L., Lin, X., 2011. SDSM: A secure data service mechanism in mobile cloud computing. In: 2011 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS). pp. 1060–1065. http://dx.doi.org/10.1109/INFCOMW.2011.5928784.

Khan, M.A., 2016. A survey of security issues for cloud computing. J. Netw. Comput. Appl. 71, 11–29. http://dx.doi.org/10.1016/j.jnca.2016.05.010.

Kishorebabu, V., Packyanathan, G., Kamatham, H., Shankar, V., 2017. An adaptive decision based interpolation scheme for the removal of high density salt and pepper noise in images. EURASIP J. Image Video Process. 2017 (1), 67. http://dx.doi.org/10.1186/s13640-017-0215-0.

Korč, F., Förstner, W., 2009. eTRIMS Image Database for Interpreting Images of Man-Made Scenes. Tech. Rep. TR-IGG-P-2009-01, Dept. of Photogrammetry, University of Bonn, http://www.ipb.uni-bonn.de/projects/etrims_db/, Accessed: Jan. 04, 2021.

Kumar, R., Goyal, R., 2019. On cloud security requirements, threats, vulnerabilities and countermeasures: A survey. Comp. Sci. Rev. 33, 1–48. http://dx.doi.org/10.1016/j.cosrev.2019.05.002.

Kumari, M., Gupta, S., Sardana, P., 2017. A survey of image encryption algorithms. 3D Res. 8 (4), 37. http://dx.doi.org/10.1007/s13319-017-0148-5.

Lahan, L., 2021. Image transformations. [Online]. Available: http://cloudinary.com/documentation/image_transformations. Accessed: Jan. 04, 2021.

Li, S.-W., Koh, J.S., Nieh, J., 2019. Protecting cloud virtual machines from hypervisor and host operating system exploits. In: 28th USENIX Security Symposium (USENIX Security 19). USENIX Association, Santa Clara, CA, pp. 1357–1374.

Li, Y., Ma, J., Miao, Y., Wang, Y., Liu, X., Choo, K.-K.R., 2020. Similarity search for encrypted images in secure cloud computing. IEEE Trans. Cloud Comput. 1. http://dx.doi.org/10.1109/TCC.2020.2989923.

Lian, S., Liu, Z., Ren, Z., Wang, H., 2007. Commutative encryption and watermarking in video compression. IEEE Trans. Circuits Syst. Video Technol. 17 (6), 774–778. http://dx.doi.org/10.1109/TCSVT.2007.896635.

Lin, E., Eskicioglu, A., Lagendijk, R., Delp, E., 2005. Advances in digital video content protection. Proc. IEEE 93 (1), 171–183. http://dx.doi.org/10.1109/JPROC.2004.839623.

Liu, Q., Sung, A.H., 2009. A new approach for JPEG resize and image splicing detection. In: Proceedings of the First ACM Workshop on Multimedia in Forensics. In: MiFor '09, Association for Computing Machinery, New York, NY, USA, ISBN: 9781605587554, pp. 43–48. http://dx.doi.org/10.1145/1631081.1631092.

Liu, F., Wang, Y., Wang, F.-C., Zhang, Y.-Z., Lin, J., 2019. Intelligent and secure content-based image retrieval for mobile users. IEEE Access 7, 119209–119222. http://dx.doi.org/10.1109/ACCESS.2019.2935222.

Liu, X., Yang, G., Mu, Y., Deng, R.H., 2020. Multi-user verifiable searchable symmetric encryption for cloud storage. IEEE Trans. Dependable Secure Comput. 17 (6), 1322–1332. http://dx.doi.org/10.1109/TDSC.2018.2876831.

Lundh, F., Clark, A., 2021. Pillow (PIL fork). [Online]. Available: https://pillow.readthedocs.io/en/stable/ Accessed: Jan. 03, 2021.

Madi, T., Jarraya, Y., Alimohammadifar, A., Majumdar, S., Wang, Y., Pourzandi, M., Wang, L., Debbabi, M., 2018. ISOTOP: Auditing virtual networks isolation across cloud layers in openstack. ACM Trans. Priv. Secur. 22 (1), http://dx.doi.org/10.1145/3267339.

Marwan, M., Kartit, A., Ouahmane, H., 2017a. Design a secure framework for cloud-based medical image storage. In: Proceedings of the 2nd International Conference on Big Data, Cloud and Applications. In: BDCA'17, Association for Computing Machinery, New York, NY, USA, http://dx.doi.org/10.1145/3090354.3090361.

Marwan, M., Kartit, A., Ouahmane, H., 2017b. Protecting medical data in cloud storage using fault-tolerance mechanism. In: Proceedings of the 2017 International Conference on Smart Digital Environment. In: ICSDE '17, Association for Computing Machinery, New York, NY, USA, pp. 214–219. http://dx.doi.org/10.1145/3128128.3128161.

McCarthy, C., Sullivan, K., Krishnan, R., 2021. Systems and methods for private cloud computing. [Online]. Available: https://patents.google.com/patent/US8495611B2/en Accessed: Jan. 04, 2021.

Nam, S.-H., Ahn, W., Mun, S.-M., Park, J., Kim, D., Yu, I.-J., Lee, H.-K., 2019. Content-aware image resizing detection using deep neural network. In: 2019 IEEE International Conference on Image Processing (ICIP). pp. 106–110. http://dx.doi.org/10.1109/ICIP.2019.8802946.

Namasudra, S., Chakraborty, R., Majumder, A., Moparthi, N.R., 2020. Securing multimedia by using DNA-based encryption in the cloud computing environment. ACM Trans. Multimedia Comput. Commun. Appl. (ISSN: 1551-6857) 16 (3s), http://dx.doi.org/10.1145/3392665.

Ni, J., Huang, Y., Luan, Z., Zhang, J., Qian, D., 2011. Virtual machine mapping policy based on load balancing in private cloud environment. In: 2011 International Conference on Cloud and Service Computing. pp. 292–295. http://dx.doi.org/10.1109/CSC.2011.6138536.

Noor, J., Akbar, H.I., Sujon, R.A., Al Islam, A.B.M.A., 2017. Secure processing-aware media storage (SPMS). In: 2017 IEEE 36th International Performance Computing and Communications Conference (IPCCC). pp. 1–8. http://dx.doi.org/10.1109/PCCC.2017.8280457.

Noor, J., Al Islam, A.A., 2017. Ibuck: Reliable and secured image processing middleware for openstack swift. In: 2017 International Conference on Networking, Systems and Security (NSysS). IEEE, pp. 144–149.

Pal, D., Vanijja, V., 2017. Effect of network qos on user qoe for a mobile video streaming service using H.265/VP9 codec. Procedia Comput. Sci. 111, 214–222. http://dx.doi.org/10.1016/j.procs.2017.06.056, The 8th International Conference on Advances in Information Technology.

Pan, W., Wang, M., Qin, J., Zhou, Z., 2021. Improved CNN-based hashing for encrypted image retrieval. Secur. Commun. Netw. (ISSN: 1939-0114) 2021, 5556634. http://dx.doi.org/10.1155/2021/5556634.

Pock, D., 2019. What is a progressive jpeg? [Online]. Available: https://www.liquidweb.com/kb/what-is-a-progressive-jpeg/ Accessed: Jan. 03, 2021,

Rad, P., Muppidi, M., Jaimes, A.S., Agaian, S.S., Jamshidi, M., 2015. A novel image encryption method to reduce decryption execution time in cloud. In: 2015 Annual IEEE Systems Conference (SysCon) Proceedings. pp. 478–482. http://dx.doi.org/10.1109/SYSCON.2015.7116797.

Rahim, N., Ahmad, J., Muhammad, K., Sangaiah, A.K., Baik, S.W., 2018. Privacy-preserving image retrieval for mobile devices with deep features on the cloud. Comput. Commun. (ISSN: 0140-3664) 127, 75–85. http://dx.doi.org/10.1016/j.comcom.2018.06.001.

Roszkowiak, L., Korzynska, A., Zak, J., Pijanowska, D., Swiderska-chadaj, Z., Markiewicz, T., 2017. Survey: Interpolation methods for whole slide image processing. J. Microsc. 265 (2), 148–158. http://dx.doi.org/10.1111/jmi.12477.

Sheidani, S., Mahmoudi-Aznaveh, A., Eslami, Z., 2021. CPA-secure privacy-preserving reversible data hiding for jpeg images. IEEE Trans. Inf. Forensics Secur. 1. http://dx.doi.org/10.1109/TIFS.2021.3080497.

Singh, P., Raman, B., Agarwal, N., Atrey, P.K., 2017. Secure cloud-based image tampering detection and localization using POB number system. ACM Trans. Multimedia Comput. Commun. Appl. 13 (3), http://dx.doi.org/10.1145/3077140.

Tak, B., Isci, C., Duri, S., Bila, N., Nadgowda, S., Doran, J., 2017. Understanding security implications of using containers in the cloud. In: 2017 USENIX Annual Technical Conference (USENIX ATC 17). USENIX Association, Santa Clara, CA, pp. 313–319.

Tanwar, V.K., Raman, B., Rajput, A.S., Bhargava, R., 2020. *Cryptolesion*: A privacy-preserving model for lesion segmentation using whale optimization over cloud. ACM Trans. Multimedia Comput. Commun. Appl. 16 (2), http://dx.doi.org/10.1145/3380743.

Toderici, G., Vincent, D., Johnston, N., Jin Hwang, S., Minnen, D., Shor, J., Covell, M., 2017. Full resolution image compression with recurrent neural networks. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR). pp. 5435–5443. http://dx.doi.org/10.1109/CVPR.2017.577.

Torres, A.D., Yan, H., Aboutalebi, A.H., Das, A., Duan, L., Rad, P., 2018. Chapter 3 - patient facial emotion recognition and sentiment analysis using secure cloud with hardware acceleration. In: Computational Intelligence for Multimedia Big Data on the Cloud with Engineering Applications. In: Intelligent Data-Centric Systems, Academic Press, pp. 61–89. http://dx.doi.org/10.1016/B978-0-12-813314-9.00003-7.

Tyleček, R., Šára, R., 2013. Spatial pattern templates for recognition of objects with regular structure. In: Pattern Recognition. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 364–374. http://dx.doi.org/10.1007/978-3-642-40602-7_39.

Tzelepi, M., Tefas, A., 2018. Deep convolutional learning for content based image retrieval. Neurocomputing (ISSN: 0925-2312) 275, 2467–2478. http://dx.doi.org/10.1016/j.neucom.2017.11.022.

Vatolin, D., Moskvin, A., Petrov, O., Trunichkin, N., 2021. MSU video quality measurement tools. [Online]. Available: http://www.compression.ru/video/quality_measure/. Accessed: Jan. 03, 2021.

Vouk, M.A., Rindos, A., Averitt, S.F., Bass, J., Bugaev, M., Kurth, A., Peeler, A., Schaffer, H.E., Sills, E.D., Stein, S., Thompson, J., Valenzisi, M., 2009. Using VCL technology to implement distributed reconfigurable data centers and computational services for educational institutions. IBM J. Res. Dev. 53 (4), 509–526. http://dx.doi.org/10.1147/JRD.2009.5429056.

Wang, Z., Bovik, A.C., 2009. Mean squared error: Love it or leave it? A new look at signal fidelity measures. IEEE Signal Process. Mag. 26 (1), 98–117. http://dx.doi.org/10.1109/MSP.2008.930649.

Wang, Z., Li, Q., 2007. Video quality assessment using a statistical model of human visual speed perception. J. Opt. Soc. Amer. 24 (12), B61–B69. http://dx.doi.org/10.1364/JOSAA.24.000B61.

Wang, X., Ma, J., Liu, X., Miao, Y., 2019. Search in my way: Practical outsourced image retrieval framework supporting unshared key. In: IEEE INFOCOM 2019 - IEEE Conference on Computer Communications. pp. 2485–2493. http://dx.doi.org/10.1109/INFOCOM.2019.8737619.

Wang, Q., Yuan, Y., 2014. Learning to resize image. Neurocomputing (ISSN: 0925-2312) 131, 357–367. http://dx.doi.org/10.1016/j.neucom.2013.10.007.

Wang, Y., Zhou, W., Zhang, P., 2017. Implementation and demonstration of qoe measurement platform. In: QoE Management in Wireless Networks. Springer International Publishing, pp. 45–57. http://dx.doi.org/10.1007/978-3-319-42454-5_5.

Wu, C.-P., Kuo, C.-C., 2005. Design of integrated multimedia compression and encryption systems. IEEE Trans. Multimed. 7 (5), 828–839. http://dx.doi.org/10.1109/TMM.2005.854469.

Xia, Z., Wang, L., Tang, J., Xiong, N.N., Weng, J., 2021. A privacy-preserving image retrieval scheme using secure local binary pattern in cloud computing. IEEE Trans. Netw. Sci. Eng. 8 (1), 318–330. http://dx.doi.org/10.1109/TNSE.2020.3038218.

Zeng, W., Lei, S., 2003. Efficient frequency domain selective scrambling of digital video. IEEE Trans. Multimed. 5 (1), 118–129. http://dx.doi.org/10.1109/TMM.2003.808817.

Zhang, Y., Xiao, X., Yang, L.-X., Xiang, Y., Zhong, S., 2020a. Secure and efficient outsourcing of PCA-based face recognition. IEEE Trans. Inf. Forensics Secur. 15, 1683–1695. http://dx.doi.org/10.1109/TIFS.2019.2947872.

Zhang, C., Zhu, L., Zhang, S., Yu, W., 2020b. TDHPPIR: An efficient deep hashing based privacy-preserving image retrieval method. Neurocomputing 406, 386–398. http://dx.doi.org/10.1016/j.neucom.2019.11.119.

Zhao, Y., Wang, W., Li, Y., Colman Meixner, C., Tornatore, M., Zhang, J., 2019. Edge computing and networking: A survey on infrastructures and applications. IEEE Access 7, 101213–101230. http://dx.doi.org/10.1109/ACCESS.2019.2927538.

Zheng, Y., Cui, H., Wang, C., Zhou, J., 2017. Privacy-preserving image denoising from external cloud databases. IEEE Trans. Inf. Forensics Secur. 12 (6), 1285–1298. http://dx.doi.org/10.1109/TIFS.2017.2656824.

Jannatun Noor has been working as a Lecturer at the Department of CSE, BRAC University since September 2018. Besides, she is working as a Graduate Research Assistant under the supervision of Prof Dr. A.B.M. Alim Al Islam in the Department of CSE at Bangladesh University of Engineering and Technology (BUET). Prior to BRACU, she worked as a TEAM LEAD of the IPV-Cloud team in IPvision Canada Inc. She received her B.Sc. and M.Sc. degrees in CSE from BUET. Her research work covers Cloud Computing, Wireless Networking, Data Mining, Big Data Analysis, Internet of Things, etc.

Saiful Islam Salim is working as a Graduate Research Assistant in the Department of CSE at Bangladesh University of Engineering and Technology (BUET) under the supervision of Prof. A. B. M. Alim Al Islam. He completed his B.Sc. from the Department of CSE at BUET. Now, he is pursuing his M.Sc. in CSE at BUET. His research interest covers Software Security, Machine Learning, Human-Computer Interaction, Image processing, Multi-Objective Optimization, Internet of Things, and Robotics.

Dr. A. B. M. Alim Al Islam is working as Professor at Bangladesh University of Engineering and Technology, Dhaka, Bangladesh. He has received B.Sc. and M.Sc. Engineering in Computer Science and Engineering from Bangladesh University of Engineering and Technology, Bangladesh. He has also received a Ph.D. in Computer Science and Engineering, 2012 from the School of ECE, Purdue University, West Lafayette in USA. His areas of research interests include Wireless Networks, Embedded Systems, Modeling & Simulation, Computer Networks Security, and Reliability Analysis.