SMC: Alternative Smart Media Compression Techniques for Edge Storage Offloading

Ali E. Elgazar
Carnegie Mellon University
aee@cs.cmu.edu

Khaled A. Harras
Carnegie Mellon University
kharras@cs.cmu.edu

Mohammad Aazam
Carnegie Mellon University
aazam@ieee.org

ABSTRACT

With the pervasiveness and growth in media technology, user-generated content has become intertwined with our day-to-day life. Such advancements, however, have enabled the exponential growth in media file sizes, which leads to shortage of storage on small-scale edge devices. While online clouds have been the default solution, they raise privacy concerns, are not fully automated, and do not adapt to different networking environments. Distributed storage systems rely on distributed file partitioning to combat concerns over privacy, and are adaptable to different networking environments. Nevertheless, such systems lack optimization via compression due to energy concerns on edge devices. In this work, we propose Smart Media Compression (SMC), a system that can be integrated with various distributed edge cloud (DEC) storage systems. SMC utilizes both a deterministic as well as a machine learning approach to classify the relevance of files to the user. Once classification is performed, SMC intelligently selects which files to compress, which files to preserve as is, and which files to offload to the distributed edge storage system. SMC dynamically adapts its parameters in order to reduce the amount of needless compression, thus minimizing energy consumption. It accomplishes this while also providing faster access to user files compared to standalone DEC systems. Our results show an improvement in average file access delay by up to 90%, while only costing an additional 14% in energy consumption.

CCS CONCEPTS

• **Information systems** → **Multimedia information systems**; • **Theory of computation** → **Data compression**; **Pattern matching**.

KEYWORDS

Compression , mobile-devices, multimedia, access-patterns, energy-efficiency

ACM Reference Format:
Ali E. Elgazar, Khaled A. Harras, and Mohammad Aazam. 2020. SMC: Alternative Smart Media Compression Techniques for Edge Storage Offloading . In *16th ACM Symposium on QoS and Security for Wireless and Mobile Networks (Q2SWinet'20), November 16–20, 2020, Alicante, Spain.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3416013.3426454

Q2SWinet'20, November 16–20, 2020, Alicante, Spain

ACM ISBN 978-1-4503-8120-8/20/11.
https://doi.org/10.1145/3416013.3426454

1 INTRODUCTION

With ongoing advances in media technologies, user generated content is exponentially growing, giving rise to media storage challenges on mobile and edge devices [1–5]. These challenges have prompted most users to resort to utilizing online clouds. However, major centralized storage clouds (CSCs) such as Apple's iCloud, Dropbox, and Google Drive, have come under fire due to multiple hacks compromising user privacy [6–8]. Given the rise of such privacy issues in CSCs, distributed edge cloud (DEC) systems have become an alternative solution to CSCs. DECs such as Sia, Symform, Beekup, and EdgeStore rely on their distributed nature on edge user devices in order to reduce cost and alleviate privacy concerns [9–12].

While the aforementioned systems promise increased privacy and cheaper additional storage, their performance can be enhanced by capitalizing on compression techniques to offer an improved user experience [13, 14]. For instance, EdgeStore is a private storage solution deployable on household devices to provide an edge-based storage cloud for users [12]. It automatically classifies and offloads unpopular files from a user's smartphone/tablet to the users' own underutilized household devices such as desktop, laptop, or even other smartphones or tablets. EdgeStore shows that most files are often not accessed by the user, and thus can offload upwards of 90% of the media files and still maintain low average access times of roughly two seconds. However, like most CSC and DEC systems, users may still access some files that were deemed unpopular and therefore have been offloaded. There is a room for improvement with respect to identifying such files, compressing them, and keeping them on the device, such that if a user accesses them in the future, the files would be quickly available. The main reason as to why these systems lack compression-based optimization is due to the fact that compression can consume large amounts of energy when performed on an edge device [15, 16].

In this work, we introduce an energy-efficient media classification and compression system, Smart Media Compression (SMC), which can be integrated with any DEC. We propose an architecture that can be integrated with DECs and centralized clouds, but focus on integration with DECs because they are more beneficial to a wider audience of users that may experience weaker network connectivity; centralized clouds typically rely on stable networking connections. SMC operates like traditional mobile storage offloading and caching solutions for DECs. Due to its easy-to-integrate architecture, it enables mobile offloading platforms that do not support caching to have lower average delays on retrieving offloaded files, by increasing local hits on a user's device. Furthermore, typical caching solutions store files as-is in limited quantity, while SMC allows many files to be retained locally via compression. This approach is typically utilized by social media platforms and messaging

applications such as Telegram and WhatsApp, and our work brings some of the techniques utilized in these platforms to DECs.

SMC has two major components, the File Popularity Classifier (FPC), and File Compression Mechanism (FCM). Employing either a deterministic or machine learning-based approach, the first component, FPC, automatically classifies files as popular and unpopular, based on user access patterns. Users typically prefer their popular files intact, and unpopular files offloaded. Our FPC further identifies a third set of files as semi-popular; these files are not frequently accessed, but based on temporal locality, may be accessed again in the near future. These semi-popular files are compressed and kept on a user's device. Overall, with such tertiary classification, we identify which files should be offloaded, compressed, or preserved as is. After the FPC classification, the second component of SMC, FCM, intelligently decides which media files to compress, and how to compress them. FCM determines if compression would benefit the user in terms of overall file access delay, based on the user's storage requirements and access patterns. This selective compression, while providing the same results in terms of file access delay, reduces overall energy consumption cost. FCM uses simple lossy FFmpeg re-encoding to reduce the size of the media files at an acceptable level of loss of quality. This allows both popular and semi-popular files to exist on the user's device, while still meeting the user's storage requirements.

We implement SMC and integrate it with a recently developed sample DEC, namely, EdgeStore [12]. We implement both SMC and EdgeStore in Java, and utilize a set of real life mobile file access traces to realistically emulate user access patterns to various media files. These realistic access patters are utilized in SMC's file classification and compression. We attach SMC to EdgeStore and compare its performance to the default EdgeStore (ES) in a metropolitan, urban, and rural environment, where network infrastructures are powerful, average, and poor, respectively. Our results show that compression is much more important when a user demands a large portion of their storage offloaded, and its impact is most beneficial in environments with relatively weak network infrastructure. SMC exhibits an improvement in file access delays by 28%, 61%, and 90% in metropolitan, urban, and rural environments, respectively. This improvement comes at an average cost of 14% energy consumption; however, without our smart FCM energy checks, such consumption would be upwards of 43% of the mobile device energy [15, 16].

2 SMART MEDIA COMPRESSION (SMC)

In this section, we discuss the architecture and functions of SMC, and how it integrates with DECs. Note that a DEC to which SMC is attached must implement any network adaptability, since SMC does not do so itself. SMC mainly provides larger benefits in terms of access delays to solutions which operate in challenged networking environments.

2.1 SMC Architecture

Fig. 1 shows the architecture of SMC and how it is integrated with a DEC. Typically, DECs have an interface that takes user input regarding which files to offload and how much of the user's storage the user wants to offload. In a DEC augmented by SMC, SMC components are inserted between the user input and the DEC interface. This enables SMC to dictate to the DEC which files should be offloaded and which should be compressed and kept based on the user's requests and media file access patterns.

SMC consists of two major components: 1) File Popularity Classifier (FPC) which utilizes user access patterns to classify files as popular/unpopular/semi-popular. 2) File Compression Mechanism (FCM) which utilizes the FPC classifications to determine which and how files are to be compressed, and which files are to be offloaded based on the amount of storage the user wishes to free from their device.

2.2 File Popularity Classifier (FPC)

We utilize FPC in order to identify which files should be offloaded, which files should be compressed, and which files should remain intact. FPC offers two methods of file classification, the first being a deterministic approach which builds upon an algorithm called Pattern Based Popularity Assessment (PBPA) [2], and the second being a machine learning approach implementing the popular Rocchio classifier [17]. Both approaches have benefits and downsides, which are discussed later on in the evaluation section.

2.2.1 Deterministic Approach. Our deterministic approach builds upon the PBPA algorithm [2] by using temporal locality to augment PBPA. PBPA classifies files over three steps.

Identifying base unpopular files: In this step, all files that have not been accessed recently (a dynamic parameter based on the user's mobile activity) are considered unpopular, and are the most favored option for offloading.

Identifying base popular files: In this step, in the set of active files, the top most accessed portion is considered popular and is favored for retaining on the user's device.

Classifying the remaining unknown files: PBPA establishes a few metrics which are utilized to gauge a user's unique access pattern to a file [2]. These metrics include the number of accesses, the intersession length (time between consecutive accesses), the spread of accesses over the file's timeline, and the length of the file's timeline itself. Using these metrics, the access patterns of the unknown files are matched to the access patterns of base popular files. Unknown files that match the base popular files are considered popular, files that do not are considered unpopular.

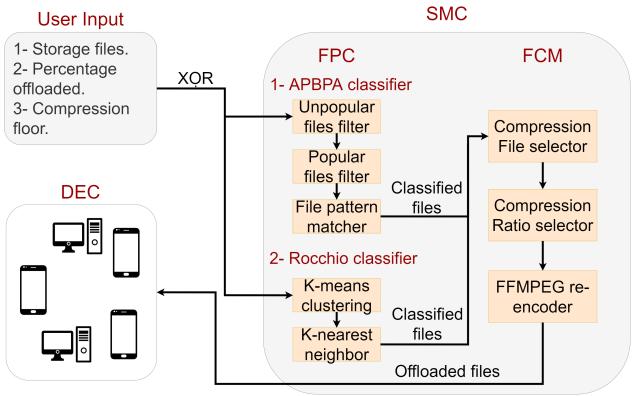


Figure 1: SMC's architecture attached to a DEC

Augmented PBPA: We create an algorithm, Augmented PBPA (APBPA), which takes advantage of file temporal locality. APBPA classifies files into three categories instead of two by modifying the final step of PBPA (classifying the remaining unknown files). In the final step of PBPA, files that do not match the access patterns of the base popular files are considered semi-popular instead of unpopular. Although semi-popular files are neither accessed frequently nor display popular access patterns, they were recently accessed and based on temporal locality they are likely to be accessed again [18]. As such, these files can be compressed and kept.

2.2.2 Machine Learning-Based Approach. In our machine learning-based approach, we implement the popular Rocchio classification algorithm [17]. Rocchio Classification is a machine learning-based approach, which relies on nearest centroid/cluster to classify items. In general, the Rocchio Classification algorithm is composed of two steps: 1- Using k-means clustering to divide a data set into k clusters. 2- Using k-nearest neighbor (kNN) algorithm on each cluster to classify new data points. We selected this approach over other more advanced machine learning approaches due to the simplicity yet effectiveness of it. These classification techniques are performed on a mobile device. While mobile devices are becoming increasingly powerful, both users and developers typically prefer to limit complex computations on mobile devices, as they can be heavily taxing on them. As such, a lightweight and effective approach such as a Rocchio classifier is best suitable for such mobile devices.

k-means clustering: k-means clustering is a vector quantization algorithm. k-means aims to divide a set of N vectors into k clusters, with each cluster having a centroid vector being the average of all vectors in said cluster. The feature vector used in our implementation is a four dimensional vector, utilizing the same metrics to measure access patterns as PBPA, and can be described as follows: $\hat{v} = (v_{ISL}, v_{Accesses}, v_{Spread}, v_{lifetime})$. In our implementation of Rocchio, we use 3-means clustering to create three clusters: popular, semi-popular, and unpopular. 3-means clustering works by starting off with three files (one for each category) as a centroid, then using euclidean distance to those three centroids, all other files are assigned a cluster. After this initial step the algorithm goes through cycles of veronoi relaxation, where a new centroid for each cluster is recalculated, and all files are reassigned based on that centroid until the algorithm converges on three clusters. Those clusters and centroids are what is used in classifying newly created files using kNN.

k-nearest neighbor: k-nearest neighbor algorithm is a vector classification algorithm. k-nearest neighbor aims to assign a vector to a class based on the nearest k vectors and their assigned classes. In our implementation, we utilize 1-nearest neighbor on the centroids of clusters produced by k-means clustering to classify newly created files. Fig. 2 shows an example of such classification. In this figure, the newly created red file has its vector's euclidean distance measured to all the clusters, and in this case, since the shortest euclidean distance as shown in the figure is to the unpopular centroid, the file would be classified as unpopular.

2.2.3 Benefits of Tertiary Classification. To measure the benefits of tertiary classification versus binary classification, we utilize real life mobile device file access traces acquired from [19]. These access traces range from 2 to 12 weeks long, with the average trace being

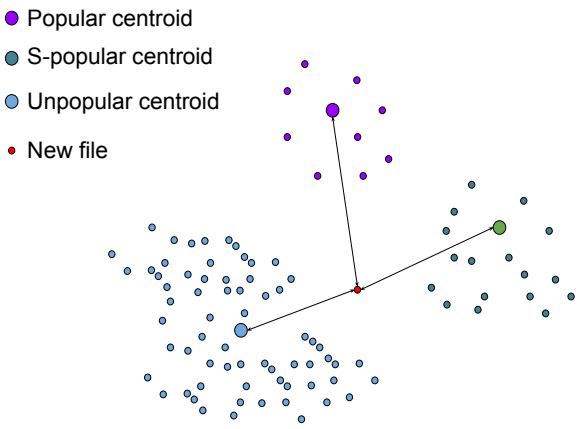


Figure 2: Classification of new files using 1-nearest neighbor on cluster centroids

8 weeks long. We classify files using both APBPA and PBPA on a snapshot of each trace at its fourth week (roughly half way through each trace). We then calculate how many files classified as unpopular using PBPA are accessed in the remaining trace, versus how many were classified as unpopular using APBPA. Using PBPA we see that 12% of files classified as unpopular are accessed, while using APBPA we see that only 4% are accessed, giving us a 67% improvement in classifying files. Files identified as semi-popular due to tertiary classification are perfect candidates for compression since they are not used frequently, but may be accessed again, thus is beneficial to keep them on the user's device.

2.3 File Compression Mechanism (FCM)

FCM is the main entity in SMC interfacing with DECs and responsible for deciding which files are compressed, how files are compressed, and which files are offloaded.

2.3.1 Compression Mechanism. In FCM, we utilize the popular library FFmpeg in order to compress media files [20]. We compared FFmpeg to other notable compression libraries such as Video Compressor used by popular messaging app Telegram [21]. We found that FFmpeg has an associated energy cost of roughly 800 Joules per 100MB of media compressed, while Video Compressor costs roughly 1730 Joules per 100MB. This cost comes at a difference in quality of the compressed files, and while Video Compressor produced higher quality compressed files as most messaging apps require, it consumed much more energy. Thus, libraries such as Video Compressor used in social media/messaging apps are not suitable for large scale compression of user data.

Note that when referring to file compression, we imply lossy compression. As such, compressed media files lose quality but are still accessible and view-able by the user without the need for decompression. Furthermore, if a file is compressed and kept on the user's device, the file has its uncompressed version offloaded such that if the user needs a high-definition version of the compressed file, they can always retrieve it.

The most important factor in compression is the compression ratio i.e., the ratio between the size of an uncompressed file, and its compressed version. In FCM, the compression ratio is dynamic

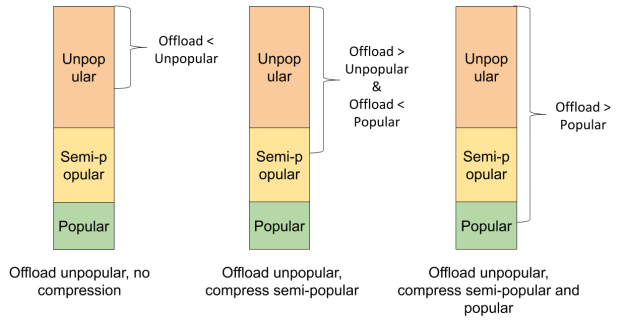


Figure 3: Visual representation of determining which files to offload and compress

based on how much of the user's storage is to be vacated, and how much is occupied by popular and semi-popular files. However, in order to calculate the compression ratio, first, we must decide on which files to compress from the popular/semi-popular files, as compressing only the semi-popular files may not be sufficient if the user wishes to offload large quantities of their storage.

2.3.2 Selecting Files to Compress. Fig. 3 provides a visual representation of how we decide which files to compress, and which files to offload. If the amount of storage space a user requests to be offloaded falls short of the size of unpopular files, then compression becomes counter-productive and consumes device energy. This occurs when the total amount of storage minus the files the user wishes offloaded (storage saved) is greater than the size of popular and semi-popular files, as such, in this case no compression should take place, set of files compressed is null. This storage requirement check allows our system to compress only when compression is needed, and thus, saves battery consumption. This is due to the fact that compressing semi-popular files in this case only yields more unpopular files being saved on the device. Thus, it minimally benefits the average access delay time, as those unpopular files now being kept are not accessed. Finally, if the amount of storage requested exceeds the amount of unpopular files, but does not exceed the amount of unpopular files combined with semi-popular files, we compress semi-popular files only. Otherwise we need to compress both semi-popular files as well as popular files.

2.3.3 File Compression Ratio. The compression ratio (CR) selection varies based on the files being compressed. If we are compressing semi-popular files only, we utilize the formula for CR1 shown below, Where TS, SR, PF, and SPF are total storage, storage requested, popular files, and semi-popular files respectively. We set the ratio of compression dynamically to match the difference between the size of popular files and storage saved, such that the compressed semi-popular files fit into that difference. On the other hand, if we must compress both popular and semi-popular files, we utilize the formula for CR2 shown below. Naturally, we would like to minimally scale popular files in comparison to semi-popular files. As such, we use α in our formula as a weight assigned a different value based on the category of file being compressed. If the file being compressed is a semi-popular file, then $\alpha = PF/SPF$, however, if the file is popular, then $\alpha = SPF/PF$.

$$\text{CR1} = \frac{\text{TS-SR-PF}}{SPF} \qquad \text{CR2} = \alpha \times \frac{\text{TS-SR}}{PF + SPF}$$

Using the compression ratio, we scale the spacial resolution of media files in order to allow said files to exist within the storage requirements. Note that temporal resolution of video and audio files are not altered. As such, the objective of FCM is to allow files that fall in the popular and semi-popular categories, regardless of their quantity, to exist on the user's device while still meeting the user's storage requirements. As such, there is no floor as to how much a file is compressed, and it is function of how much storage the user requests.

3 EVALUATION

In this section, we discuss our evaluation setup and present our results. To show the impact of Smart Media Compression on DECs, we attach SMC to a recently developed DEC, EdgeStore, and we compare the results between default EdgeStore (ES), and SMC integrated with EdgeStore (SMC).

3.1 EdgeStore Overview

EdgeStore is a private, deployable DEC platform [12]. EdgeStore allows users to utilize their household compute devices as a private distributed storage cloud where they can automatically offload files, taking advantage of the underutilized storage most users have between their household devices. EdgeStore supports automatic offloading of user files based on user access patterns, as such, it is a suitable candidate for evaluating SMC, since SMC also relies on user access patterns for file compression.

3.2 Experimental Setup

We implement EdgeStore and SMC in Java, and utilize the Java-based ONE simulator to simulate device communication in a household scenario [22]. The experiment is based on 90 days of in-simulation time, in which a main mobile device access files. EdgeStore automatically offloads files it deems unpopular during the simulation, while SMC+EdgeStore compresses some files and offloads the others.

In order to account for users experiencing different network conditions, we create three separate simulation environments addressing different networking infrastructures (rural, urban, and metropolitan). In the rural environment, devices communicate using only direct techniques (Bluetooth, ad-hoc), while in the urban environment devices communicate using WiFi only. In the metropolitan environment devices utilize WiFi when it is available and 4G network otherwise.

As EdgeStore is a storage solution that utilizes household edge devices, we utilize household mobility models to simulate user node movement. We utilize the same mobility model utilized by previous EdgeStore evaluation [12]. It should be noted that if users offload a file and later lose connection to the device holding that file, they must wait till the connection is re-established to retrieve the file. Household mobility models generally consist of a number of hours of communication uptime between nodes, followed by a number of hours of communication downtime. This behavior follows the principle that when a node is in its house, it can communicate with the other household devices. When nodes leave the house, they

are occasionally randomly paired with other nodes from the house. This simulates familial behavior in traveling together. Our mobility model is based on the observations of various works in studying the mobility of household members, as well as the mobility of members in ad hoc networks in general [23][24][25].

3.3 Dataset

To simulate user accesses of files, we utilize the same set of real life mobile device file access traces utilized in EdgeStore evaluation [19]. The user traces have been filtered to contain only audio, video, and image files that are larger than a particular threshold in size (2MB for audio and video, and 500KB for images). This filtering is done to ensure that the files left in the traces are the ones accessed by users, and not by the system itself (logo images, device sound clips, etc.). In this evaluation, we use the most active data trace from all the mobile data access traces we acquired. This active data trace contains 1417, 181, 18 images, videos, and audios, respectively, with a combined number of 112417 accesses to said files over the 90-day trace. The average sizes of images, videos, and audios, are 2.1, 91.4, and 5.4MBs, respectively. During the 90-day simulation, an offloading device is fitted with an access trace, and the offloading/compression decisions are based on the file access patterns for that user.

3.4 Parameters and Metrics

In order to gauage the impact of our parameters individually, as we vary a parameter, other parameters are kept at a default value. We utilize four parameters and three metrics in our evaluations:

Percent of storage offloaded: This percentage represents the amount of storage to be offloaded, and is typically input by the user to SMC. We vary this percentage between 20%, 50% and 90% with a default value of 90%.

Compression floor: This parameter represents the minimum quality that a file can be compressed to. For example, if the compression floor is set to 480p, files above that quality can not be compressed to under 480p. This parameter can be set to any value between floorless, 144p, and 360p with a default value of floorless.

Classification Algorithm: We compare our two file classifiers APBPA and Rocchio in order to showcase the benefits and downsides of deterministic and machine learning classification of user files on our system. The default classifier for our evaluation is APBPA

Learning period: This period of time is given to both classification algorithms to learn from the users' access patterns. This parameter is varied between 2, 3, and 4 weeks with a default value of 2 weeks.

Access Delay: This is the time taken for a file to be presented to the user upon request. A file access request can either be a local hit (available on the native device) or a miss (offloaded to another edge device). Compression plays a huge factor in access delays as more files are available on the user's system instead of being offloaded.

Energy Consumption: Energy consumption is impacted by the number and size of files compressed, as well as the communication technology utilized. In our evaluations, we fix the throughput of our 4G, WiFi, and Bluetooth interfaces to 75 Mbps, 40 Mbps, and 20 Mbps respectively [26]. Our tests showed that each 1MB of data transferred consumes on average 25, 15, and 10 Joules when utilizing 4G, WiFi, and Bluetooth respectively. APBPA computations roughly consume 550 Joules per hour. Compression using FFmpeg consumes roughly 800 Joules per 100MB of media. These numbers are based on tests performed on an HTC Android device with a 2600 mAh battery; at 5V, this is roughly 46800 Joules of energy. These values were used to simulate battery consumption throughout the experiment. We note that when users view a compressed file, they always opt to retrieve the original HD version of the file, causing additional energy usage.

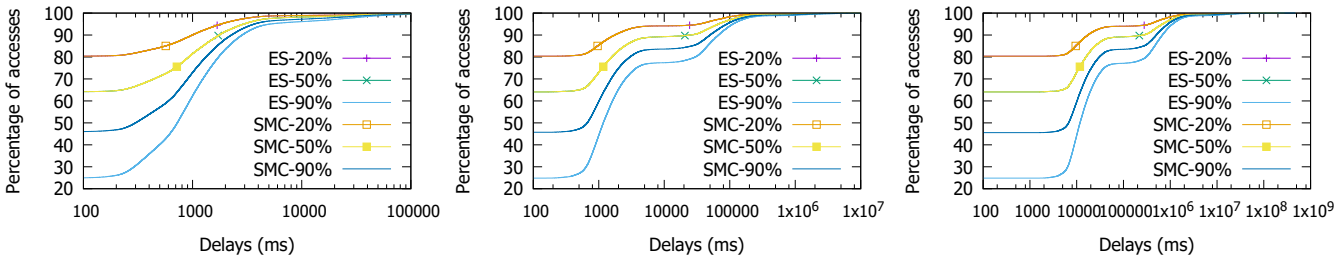
User Mean Opinion Score (MOS): This metric represents an approximated user opinion score based on previous studies into Quality of Experience for users viewing media on their mobile devices.

3.5 Results

3.5.1 Impact of smart compression checks. Fig. 4 shows the CDF of delays in every environment given different percentage of storage offloaded. Note that the X-axis starts at 100ms, as the minimum non-zero delay on accessing any file in the simulation is higher than 100ms. In all three environments, there is no improvement in access delays between ES and SMC when we offload 20% and 50% of files, giving us overlapping lines between ES-20%/SMC-20%, and ES-50%/SMC-50%. This is due to the preventative compression checks implemented in FCM. In the trace, 62% of the files were unpopular on average. As such, when the user chooses to offload less than 62% of files, compression does not benefit the system, as compressing files merely adds more unpopular files to the system which are unlikely to be accessed. In this case, SMC's compression checks prevent wasteful compression and no files are compressed, giving the same result as ES.

3.5.2 Impact of compression on access delays. When the user offloads 90% of their storage, delay increases significantly as more files are offloaded and thus, need to be retrieved later. However, using SMC, the percentage of instantaneous accesses falls around 47%, compared to 25% when using ES. This increase in instantaneous access causes the average delay to plummet. In the metropolitan, urban, and rural environments, the average delay drops from 312ms to 243ms, 432ms to 267 ms, and 2.1s to 1.1s respectively, giving us a 28%, 61%, and 90% improvement in delays on average, respectively. These results show that while SMC provides an improvement to storage solutions in any environment, SMC is most beneficial in environments with low bandwidth and connectivity, where it is more difficult to retrieve offloaded files, and thus more important to keep them on a user's device.

The improvements from ES to SMC are due to the fact that with tertiary classification and compression, the user's device offloads less files. Those improvements are amplified in urban and rural environments due to connection downtimes. This causes devices using ES to have to wait prolonged periods of times until they can retrieve files, in comparison to SMC where those files were classified as semi-popular and compressed instead. This shows that in the absence of a stable network connection, compressing files and keeping them on a user's device becomes imperative, as waiting for a connection to retrieve a file causes the average delays on accessing a file to skyrocket.



(a) Metropolitan environment (b) Urban environment (c) Rural environment

Figure 4: CDF of access delays in every environment using ES and SMC with varying percent of files offloaded

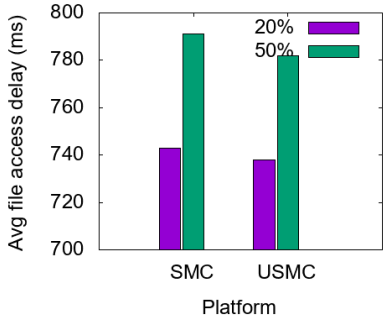


Figure 5: Access delay in SMC versus Unchecked SMC (USMC) with varying offload percentages

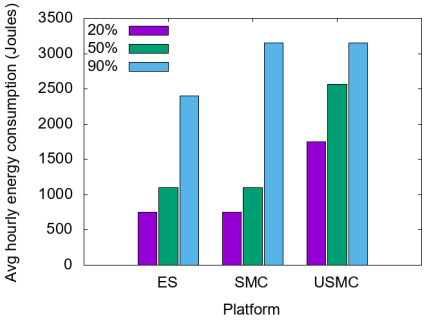


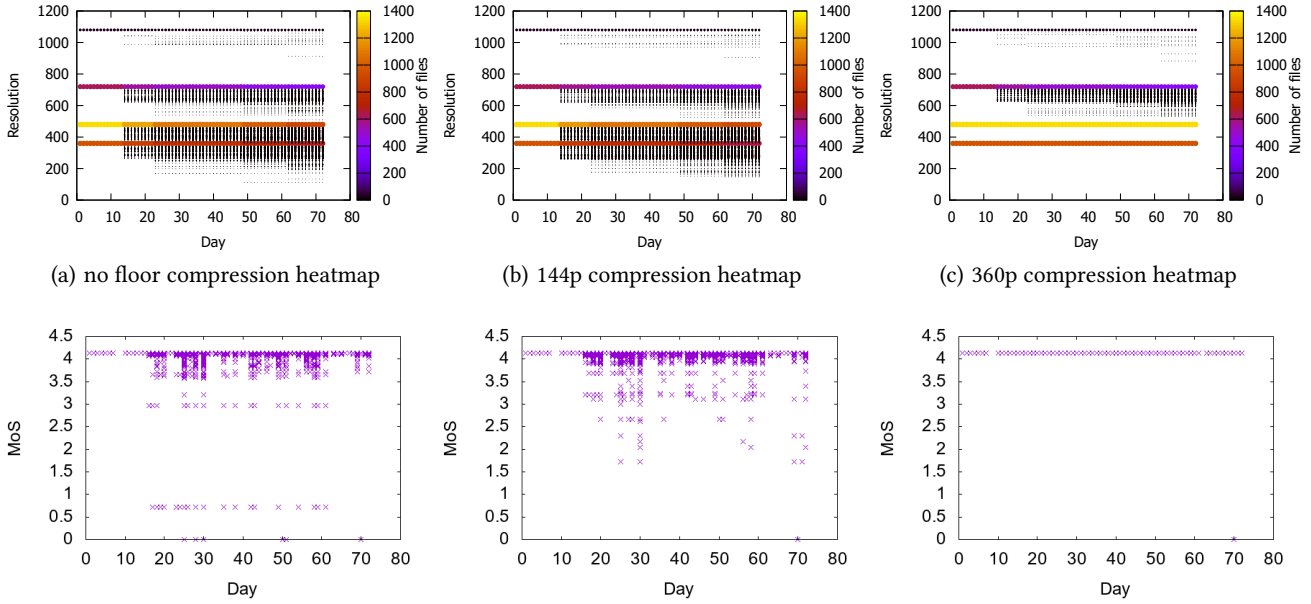
Figure 6: Comparison between ES, SMC, and USMC in energy consumption with varying offload percentages

3.5.3 Impact of unchecked compression on average delays. To showcase the importance of smart compression, we disable the aforementioned checks in a few of our simulations and compare the results using the rural environment. Fig. 5 compares SMC and Unchecked SMC (USMC) with compression checks turned off. We observe little benefit in USMC compared to SMC, with USMC producing an average delay only 0.6 and 1.1% improved when we offload 20 and 50% of files. Not only is the improvement minor without compression checks, but we consume more energy as the device is blindly compressing files without need.

3.5.4 Impact of unchecked compression on battery usage. Fig. 6 shows the comparison between ES, SMC, and USMC in terms of energy consumption. Energy drain is bound to occur in our system whether ES or SMC is utilized, due to the large quantities of files being offloaded. During the simulation, there is no difference between ES and SMC when we offload less than 50% of files in terms of energy consumption. In comparison, USMC has a much higher energy consumption for the same amount of files offloaded. As we can see, when we offload an amount of files more than the amount of unpopular files (90%), the energy consumption of SMC is the same as USMC as the checks only apply to the aforementioned specific case. The average hourly energy consumption in all simulations for ES, SMC, and USMC is 1415, 1660, and 2490 Joules respectively. As such, devices running ES, SMC, and USMC ran out of battery every 33, 28, and 19 hours respectively. While SMC costs an additional 14% of energy consumption, it prevents an additional 43% energy consumption through its smart checks. For comparison, on the same device on which these energy calculations took place, an hour of browsing the Internet with the display brightness at 50% consumes roughly 3600 Joules.

3.5.5 Impact of compression floor on MOS. Fig. 7 shows a file quality heatmap over time for different compression floors. The width of a spot as well as the colour of a spot on this heatmap changes as more files are aggregated in that spot. Fig. 7 also shows a MOS score per user access for matching compression floors. As we decrease compression floors, the overall quality of files continuously drops. However, we only see drops in MOS once we go below 360p compression floor. This is due to the fact that typical mobile devices have smaller screens of around 360p width, as such, resolutions higher than 360p seem comparable on the typical mobile device [27, 28]. Naturally, as we increase the compression floor, file access delays are increased as less files are compressed and kept on the user's device, causing the device to retrieve offloaded files that are later accessed. This shows that with a proper compression floor set by the user based on his screen size, we can effectively compress files to reduce file access delays while avoiding incurring a cost in terms of MOS.

3.5.6 Comparison between deterministic and machine learning approaches. We compare APBPA to Rocchio classifier given three different lengths of learning period. We create two versions of SMC: 1- APBPA-SMC. and 2- Rocchio-SMC, and we simulate both versions in the Rural environment with 90% of files offloaded. Fig. 8 shows



(a) no floor compression heatmap (b) 144p compression heatmap (c) 360p compression heatmap

(d) no floor compression user MOS (e) 144p compression user MOS (f) 360p compression user MOS

Figure 7: File quality heatmaps and user MOS of compressing files using no floor, 144p, and 360p floors

this comparison. As we extend the learning period for APBPA, the increase in performance is negligible. However, the increase in performance for Rocchio is more substantial and as the learning period increases, it seems that Rocchio's results converge to that of APBPA. This can be explained by examining the percentage of classifications for each class of files given different weeks.

Fig. 9 shows the percentage of classification as different training periods are utilized. We see that the Rocchio classifier approaches similar classification to APBPA as the training period grows, with the initial 2 week period (R-2W) being the most stringent in its classification. Since Rocchio classifies more files as unpopular in R-2W, it offloads more files and must retrieve more files, leading to longer access delays.

While the machine learning approach provides longer access delays, it has the benefit of providing better MOS. Fig. 10 shows the MOS when performing floorless compression and utilizing the Rocchio classifier. In comparison, Fig. 7d shows the MOS For floorless compression while utilizing APBPA. We can see that Rocchio provides better MOS than APBPA, and this is mainly due to the fact that Rocchio compresses less files and offloads more files.

4 RELATED WORK

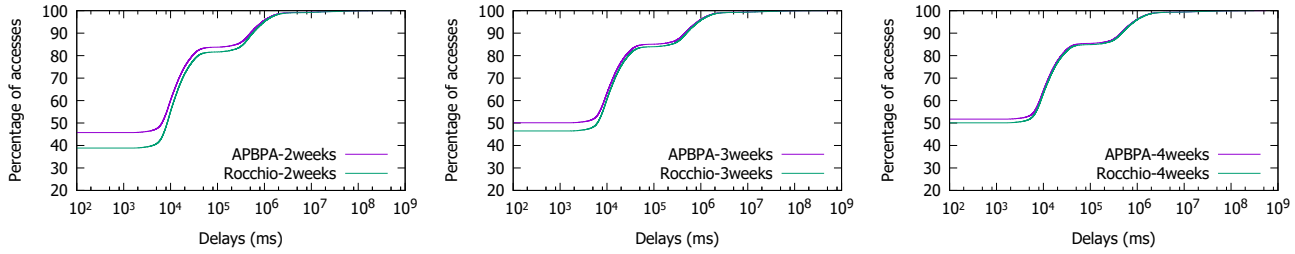
The goal of Edge/Fog computing is to mainly improve the performance of applications such as smart homes [29–31], mhealth [32, 33], vision-based systems, and augmented reality systems [34, 35], that demand higher compute resources, but cannot tolerate cloud latency. Thus, contributions in this domain have primarily focused on task and computational offloading solutions [36–40]. This work, however, focuses on storage-based challenges in these Edge environments. We therefore focus our related work on edge storage systems and compression in edge networks.

4.1 Edge Storage Systems

Early work in the field, such as Symform [10] succeeded in providing a worldwide distributed storage platform. This platform provided incentives for users to volunteer their unused devices as storage caches, which users from across the world could utilize. Unfortunately, as of 2016, Symform has officially been shut down, leading many of their patrons to seek alternatives.

Following that, the Sia platform [9] now provides an online, decentralized cloud platform, where user files are broken apart, encrypted, and offloaded to many storage caches all over the world. This approach makes it extremely difficult for malicious hackers to compromise the safety of a user's file. Other work such as Rizzo et al. propose a storage offloading system called "Beekup", based on Tahoe-LAFS, a distributed file system [41]. Beekup was designed as a distributed offloading solution very similar to Sia. Although Sia and Beekup are both excellent examples of well-made distributed storage solutions, neither platforms support any ease-of-use features such as automatic offloading, or capitalize on data gathered from user access patterns. These platforms merely provide a storage dump for files to users.

More recent work, examined the potential of automating user file offloading through an edge storage offloading system called EdgeStore [12]. Utilizing a file popularity classification algorithm, EdgeStore classified files in a binary fashion as either popular or unpopular based on the file usage patterns, and unpopular files were automatically offloaded to underutilized devices at the user's



(a) 2 week learning period (b) 3 week learning period (c) 4 week learning period

Figure 8: CDF of access delays comparing SMC with APBPA, versus SMC with Rocchio classifications.

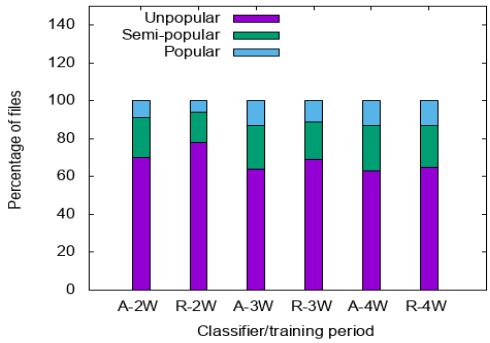


Figure 9: APBPA (A-XW) versus Rocchio (R-XW) classification percentage given different weeks.

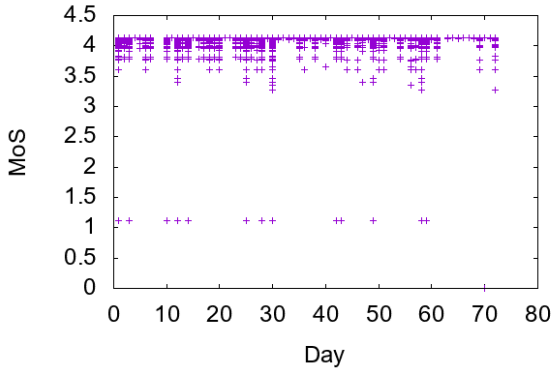


Figure 10: User MOS with Rocchio classification

home. Our work however, determines that a binary classification system is not sufficient, and that such automated systems can benefit tremendously from a tertiary classification scheme. Such schemes allow files which are not exactly popular for the user, but are likely to be used again, to be kept on the user's system in a compressed state.

4.2 Compression in Edge Networking

Compression in order to reduce network traffic is no novel idea. Work in the early 2000s discussed different techniques in compressing media files in order to reduce network traffic. For example, Arici et. al. discuss PINCO, a pipelined scheme for compressing data as it is transmitted from wireless distributed sensor networks [42].

More recently, as the definition of "Edge" became less opaque, research began re-visiting old compression techniques and applying them to complex topics such as deep learning at the edge [15]. However, regardless of the applied topic, the object of such work has almost always been reducing network traffic at the edge, and typically revolved around caching data chunks at nearby edge devices for fast and low-latency access [43, 44]. Our work is distinct in that our focus is to automatically compress files on a user's system, in order to increase local hits when accessing files on a user's device. In that spirit, our work combines file classification, automated offloading, and cache-like local compression into a platform that can be easily integrated with any offloading system to provide it with these benefits.

5 CONCLUSION AND FUTURE WORK

In this paper, we have argued the need for a private and automated edge storage systems, and how intelligent automated file classification and compression can benefit such systems. We proposed a distributed edge cloud integratable system, SMC, which aids in automated offloading and compression of media files and smart devices. We showed how tertiary file popularity classification can be more advantageous than binary. We have also demonstrated how SMC accounts for dynamic factors, such as the amount of storage the user requests offloaded and the ratio between popular and semi-popular files in the user's system. Lastly, we have examined the impact of such compression techniques on user mean opinion scores, and we showed how up to a certain degree, there is no noticeable impact on user opinion from compression, due to the nature of viewing media files on mobile devices which have smaller screen sizes.

For future work, we intend to examine pre-fetching techniques to reduce the need for compression, thus reducing energy waste. Additionally, we will investigate the possibility of having variable compression ratios based on the popularity of the files, instead of treating files in the same category as equals. For example, we intend to make it such that every single file in the system has its own compression ratio whether or not it's popular or unpopular. We also plan to more extensively evaluate the system potentially through a real deployment with real users at a decent scale.

REFERENCES

[1] G. Shao, "Understanding the appeal of user-generated media: a uses and gratification perspective," *Internet Research*, vol. 19, no. 1, pp. 7–25, 2009.

[2] A. Elgazar, K. Harras, M. Aazam, and A. Mtibaa, "Towards intelligent edge storage management: Determining and predicting mobile file popularity," in *2018 MobileCloud*. IEEE, 2018, pp. 23–28.

[3] A. Saeed, M. Ammar, K. A. Harras, and E. Zegura, "Vision: The case for symbiosis in the internet of things," in *Proceedings of the 6th International Workshop on Mobile Cloud Computing and Services*. ACM, 2015, pp. 23–27.

[4] M. Ibrahim, M. Gruteser, K. A. Harras, and M. Youssef, "Over-the-air tv detection using mobile devices," in *IEEE ICCCN*, 2017, pp. 1–9.

[5] A. Saeed, A. Abdelkader, M. Khan, A. Neishaboori, K. A. Harras, and A. Mohamed, "Argus: realistic target coverage by drones," in *ACM/IEEE IPSN*, 2017.

[6] "Dropbox hacking," https://tinyurl.com/jzptkee.

[7] "Apple hacking gets worse," http://www.zdnet.com/article/icloud-accounts-breach-gets-bigger-here-is-what-we-know/.

[8] "Google cloud hacking," http://searchengineland.com/after-the-googlehack-33508.

[9] D. Vorick and L. Champine, "Sia: Simple decentralized storage," Technical Report, Sia, 2014, Tech. Rep., 2014.

[10] Y.-Y. Teing, A. Dehghantanha, K.-K. R. Choo, T. Dargahi, and M. Conti, "Forensic investigation of cooperative storage cloud service: symform as a case study," *Journal of Forensic Sciences*, 2016.

[11] F. e. a. Rizzo, "Beekup: A distributed and safe p2p storage framework for ioe applications," in *ICIN 2017*. IEEE, 2017, pp. 44–51.

[12] A. Elgazar, M. Aazam, and K. Harras, "Edgestore: Leveraging edge devices for mobile storage offloading," in *IEEE CloudCom*, 2018, pp. 56–61.

[13] C. M. Sadler and M. Martonosi, "Data compression algorithms for energy-constrained devices in delay tolerant networks," in *Embedded networked sensor systems*. ACM, 2006, pp. 265–278.

[14] T. Ma, M. Hempel, D. Peng, and H. Sharif, "A survey of energy-efficient compression and communication techniques for multimedia in resource constrained systems," *IEEE Communications Surveys & Tutorials*, vol. 15, no. 3, pp. 963–972, 2013.

[15] C. Hardy, E. Le Merrer, and B. Sericola, "Distributed deep learning on edge-devices: feasibility via adaptive compression," in *2017 NCA*. IEEE, 2017, pp. 1–8.

[16] J. Azar, A. Makhoul, M. Barhamgi, and R. Couturier, "An energy efficient iot data compression approach for edge machine learning," *FGCS*, vol. 96, pp. 168–175, 2019.

[17] Rocchio classification. Retrieved on (2019-9-12). [Online]. Available: https://nlp.stanford.edu/IR-book/html/htmledition/rocchio-classification-1.html

[18] S. Jiang, X. Ding, F. Chen, E. Tan, and X. Zhang, "Dulo: an effective buffer cache management scheme to exploit both temporal and spatial locality," in *4th conference on USENIX File and Storage Technologies*, vol. 4, 2005, pp. 8–8.

[19] R. Friedman and D. Sainz, "File system usage in android mobile phones," in *9th SSC*. ACM, 2016, p. 16.

[20] Ffmpeg. Retrieved on (2019-9-12). [Online]. Available: https://www.ffmpeg.org/

[21] Telegram video compressor. Retrieved on (2019-9-12). [Online]. Available: https://github.com/lalongooo/VideoCompressor

[22] A. Keränen, J. Ott, and T. Kärkkäinen, "The one simulator for dtn protocol evaluation," in *STT*. ICST, 2009, p. 55.

[23] V. A. Davies *et al.*, "Evaluating mobility models within an ad hoc network," Master's thesis, Citeseer, 2000.

[24] J. P. Gliebe and F. S. Koppelman, "A model of joint activity participation between household members," *Transportation*, vol. 29, no. 1, pp. 49–72, 2002.

[25] T. F. Golob, "A model of household demand for activity participation and mobility," 1996.

[26] Throughput comparison. Retrieved on (2019-9-12). [Online]. Available: http://www.bandwidthplace.com/internet-speed-test-3g-4g-lte-and-wifi-who-wins-article/

[27] M. H. Jofri, M. F. M. Fudzee, M. N. Ismail, S. Kasim, and J. Abawajy, "Quality of experience (qoe) aware video attributes determination for mobile streaming using hybrid profiling," Ph.D. dissertation, Universiti Tun Hussein Onn Malaysia, 2016.

[28] Mobile resolutions. Retrieved on (2020-9-12). [Online]. Available: https://gs.statcounter.com/screen-resolution-stats/mobile/worldwide

[29] H. Abdelnasser, K. Harras, and M. Youssef, "A ubiquitous wifi-based fine-grained gesture recognition system," *IEEE Transactions on Mobile Computing*, vol. 18, no. 11, pp. 2474–2487, 2018.

[30] H. Abdelnasser, K. A. Harras, and M. Youssef, "Magstroke: A magnetic based virtual keyboard for off-the-shelf smart devices," in *IEEE SECON*, 2020, pp. 1–9.

[31] M. A. Shah, K. A. Harras, and B. Raj, "Sherlock: A crowd-sourced system for automatic tagging of indoor floor plans," in *IEEE MASS*, 2020.

[32] M. F. Al-Sa'D, M. Tlili, A. A. Abdellatif, A. Mohamed, T. Elfouly, K. Harras, M. D. O'Connor *et al.*, "A deep learning approach for vital signs compression and energy efficient delivery in mhealth systems," *IEEE Access*, vol. 6, pp. 33 727–33 739, 2018.

[33] A. Emam, A. A. Abdellatif, A. Mohamed, and K. A. Harras, "Edgehealth: An energy-efficient edge-based remote mhealth monitoring system," in *IEEE WCNC*, 2019, pp. 1–7.

[34] A. Saeed, A. Abdelkader, M. Khan, A. Neishaboori, K. A. Harras, and A. Mohamed, "On realistic target coverage by autonomous drones," *ACM Transactions on Sensor Networks (TOSN)*, vol. 15, no. 3, pp. 1–33, 2019.

[35] A. Saeed, M. Ammar, E. Zegura, and K. Harras, "If you can't beat them, augment them: Improving local wifi with only above-driver changes," in *IEEE ICNP*, 2018.

[36] K. Habak, E. W. Zegura, M. Ammar, and K. A. Harras, "Workload management for dynamic mobile device clusters in edge femtoclouds," in *ACM/IEEE SEC*, 2017, pp. 1–14.

[37] K. Habak, M. Ammar, K. A. Harras, and E. Zegura, "Femto clouds: Leveraging mobile devices to provide cloud service at the edge," in *IEEE CLOUD*, 2015, pp. 9–16.

[38] K. Habak, C. Shi, E. W. Zegura, K. A. Harras, and M. Ammar, "Elastic mobile device clouds: Leveraging mobile devices to provide cloud computing services at the edge," *Fog for 5G and IoT*, p. 159, 2017.

[39] H. K. Gedawy, K. Habak, K. Harras, and M. Hamdi, "Ramos: A resource-aware multi-objective system for edge computing," *IEEE Transactions on Mobile Computing*, 2020.

[40] H. Gedawy, K. A. Harras, K. Habak, and M. Hamdi, "Femtoclouds beyond the edge: The overlooked data centers," *IEEE Internet of Things Magazine*, vol. 3, no. 1, pp. 44–49, 2020.

[41] Tahoe-lafs. Retrieved on (2019-9-12). [Online]. Available: https://en.wikipedia.org/wiki/Tahoe-LAFS

[42] T. Arici, B. Gedik, Y. Altunbasak, and L. Liu, "Pinco: A pipelined in-network compression scheme for data collection in wireless sensor networks," in *2003 IEEE CCN*. IEEE, 2003, pp. 539–544.

[43] L. Liu, X. Chen, Z. Lu, L. Wang, and X. Wen, "Mobile-edge computing framework with data compression for wireless network in energy internet," *Tsinghua Science and Technology*, vol. 24, no. 3, 2019.

[44] V. Aggarwal, Y.-F. R. Chen, T. Lan, and Y. Xiang, "Sprout: A functional caching approach to minimize service latency in erasure-coded storage," *IEEE/ACM Transactions on Networking*, vol. 25, no. 6, 2017.