A Framework for Optimizing File Transfers between Mobile Devices and the Cloud

Armen Dzhagaryan, Aleksandar Milenković
Electrical and Computer Engineering
The University of Alabama in Huntsville
Huntsville, AL, USA

***Abstract*—An exponential growth of data traffic that originates on mobile devices and a shift toward cloud computing necessitate finding new approaches to optimize file transfers. Whereas compression utilities can improve effective throughput of file transfers between mobile devices and the cloud, finding the best-performing utility for a given file transfer is a challenging task. In this paper we introduce a framework for optimizing file transfers that relies on agents running on both mobile devices and the cloud. The agents are responsible to select an effective transfer mode by considering characteristics of files to be transferred, network conditions, and mobile device performance. The framework is implemented and experimentally evaluated for file uploads and downloads initiated from a smartphone, while varying WLAN network conditions. The results of the evaluation show that the framework effectively increases upload throughputs in range from 1.46 to 2.38 times relative to uncompressed uploads and from 1.02 to 2.97 times relative to default compressed uploads. Similarly, it improves download throughputs in range of 1.47 to 2.5 times relative to uncompressed downloads and up to 1.2 times relative to default compressed downloads.**

I. Introduction

Cloud and mobile computing represent two emerging trends in modern computing and communication. With a rapid shift toward cloud-based computing, mobile devices have become the dominant platforms for consuming and generating digital information [1]. Mobile devices rely on cloud services for computing and storage needs as they are typically constrained in processing power, storage capacity, energy, and communication bandwidth. Data originating on mobile devices (e.g., physiological and inertial data from health monitors, videos, images, documents, messages) are transferred to the cloud. Similarly, data stored in the cloud (e.g., documents, applications, maps, messages, commands) are transferred from the cloud to mobile devices. These trends result in an exponential growth of global mobile data traffic that reached 7.2 exabytes per month in 2016 and is forecast to increase 7 times from 2016 to 2021 [2]. It is thus critical to guarantee fast, low-latency, and high-throughput communication between mobile devices and the cloud.

Lossless data compression is currently being used to reduce the required bandwidth during web page loads and downloads of files and applications. Google's Flywheel proxy [3], Google Chrome, and Amazon Silk use proxy servers to provide HTTP compression. For file downloads, Google services, such as Gmail and Drive, provide zip compression of files and attachments. Application stores, e.g., Google Play and Apple's App Store, use zip-derived containers for application distribution, while Linux software repositories use compression utilities such as *gzip*, *bzip2*, and *xz*.

In this paper, we introduce a framework for optimizing data file transfers between mobile devices and the cloud by utilizing compression utilities. The framework seamlessly selects a file transfer mode that maximizes the effective throughput - the file can be transferred uncompressed or using any compression (utility, level) pair from a set of available options. The framework selects an effective transfer mode by considering (i) characteristics of an input file, (ii) characteristics of network connection, (iii) analytical models describing effective throughputs, and (iv) characteristics of the device that initiates the transfer. The effectiveness of the proposed framework is experimentally evaluated using a smartphone that performs upload and download file transfers for a range of input files and network conditions. The performance of the framework is compared to the performance of uncompressed and the default compressed file transfers that use *gzip* with *-6* compression level. The framework improves effective upload throughputs from 1.46 to 2.38 times relative to uncompressed uploads and from 1.02 to 2.97 times relative to default compressed uploads. It improves effective download throughput from 1.47 to 2.5 times relative to uncompressed downloads and up to 1.2 times relative to default compressed downloads.

Section II presents background and motivation. Section III gives an overview of the proposed framework and describes its components. Section IV describes experimental evaluation. Section V presents the results of the evaluation. Section VI summarizes findings and concludes the paper.

II. Background and Motivation

Fig. 1 illustrates data transfers initiated from a mobile device. A data file can be uploaded to the cloud or downloaded from the cloud uncompressed or compressed.

978-1-5386-3531-5/17/$31.00

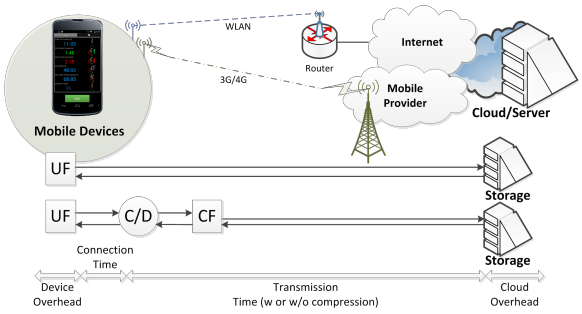

Fig. 1. Data transfers between mobile devices and the cloud.

TABLE I
COMPRESSION UTILITIES

Utility	Levels	Version	Notes
gzip	1-9 (6)	1.6	DEFLATE (Ziv-Lempel, Huffman)
lzop	1-9 (6)	1.03	LZO (Lempel-Ziv-Oberhumer)
bzip2	1-9 (6)	1.0.6	RLE+BWT+MTF+Huffman
xz	0-9 (6)	5.1.0a	LZMA2
pigz	1-9 (6)	2.3	Parallel implementation of gzip
pbzip2	1-9 (9)	1.1.12	Parallel implementation of bzip2

For uncompressed transfers, an uncompressed file (*UF*) is uploaded or downloaded over a network directly. For compressed uploads, the uncompressed file is first compressed locally on to a mobile device, and then a compressed file (*CF*) is uploaded to the cloud over the network. For compressed downloads, a compressed version of the requested file is downloaded from the cloud, and then the compressed file is decompressed locally. The file transfer and local (de)compression tasks are often overlapped in time using piping. Compressed uploads and downloads utilize one of available compression utilities. Each compression utility supports a range of compression levels that allow us to trade off speed for compression ratio (*CR*): lower levels - speed, higher levels - better compression ratio.

We consider six common compression utilities listed in Table I. *gzip* and *lzop* utilities are relatively fast, whereas *bzip2* and *xz* provide a higher compression ratio. *pigz* and *pbzip2* are parallel versions of *gzip* and *bzip2*, respectively. They exploit multiple processor cores in modern mobile devices to speed up (de)compression tasks. For each, we consider at least three compression levels: L - low (1), M - medium (6), and H - high (9).

A. Related Work

Recent measurement-based studies [4]–[7] showed that compressed transfers over WLAN and cellular interfaces outperform corresponding uncompressed file transfers. These studies showed that not a single combination of a compression utility and level performs the best for all file transfers and network conditions.

To optimize file transfers, several studies introduced run-time techniques for deciding whether to use compressed transfer or not [8]–[10]. In most closely related work to ours [8], 32 KB blocks of streaming data are analyzed at network stack level to determine which compression technique to apply, if any. Instead of analyzing data in-transit, other studies used pre-compression techniques to estimate compression ratios for selecting the optimal transfer method [9], [10]. Pre-compression is done either on the file header or multiple segments within a file prior to the file transfer. While sharing several aspects, these studies use a limited selection of compression utilities (*gzip*, *bzip2*, and *lzo*) and rely on time and energy consuming pre-compression of files, or on frequent analysis of data blocks at the network level. In addition, studies [9], [10] rely on compression ratio as the sole factor in selecting an optimal transfer mode, which may result in suboptimal effective throughput.

B. The Case of Intelligent File Transfers

To illustrate challenges in selecting an optimal transfer mode, we conduct a measurement-based evaluation of the effectiveness of uncompressed and compressed transfers to and from a remote server initiated on the OnePlus One smartphone connected to the Internet over its WLAN interface. To measure the effectiveness of file transfers we use the effective throughput (*Th*), defined as the ratio between the uncompressed file size in megabytes and the time needed to complete the file transfer. This metric captures the system's ability to perform a file transfer with the least amount of time regardless of a transfer mode. To demonstrate the impact of network connection parameters, the measurements are performed when the WLAN network throughput is set to 0.5 MB/s and 5 MB/s. The measurements for each transfer are repeated three times to mask variability of network conditions. A detailed description of measurement setup can be found in [6], [11].

Upload examples. Two text files containing information gathered on a wearable health monitor are uploaded to the cloud using uncompressed and compressed file transfers. The first file, BRW.csv (19.7 MB), contains raw samples from a breathing sensor. The second file, LOG.csv (4.7 MB), contains periodic health logs. These types of files are often uploaded to the cloud-based health monitoring applications. Fig. 2 shows compression ratio (*CR*), the uncompressed upload throughput (*Th.UUP*), and the compressed upload throughput (*Th.CUP*) for uploads over 0.5 MB/s and 5 MB/s networks. For 0.5 MB/s network, *Th.UUP* matches the network throughput (*Th.UP*). For BRW.csv, the best compressed upload using *xz -1* improves the effective throughput 10.33 times (5.45 MB/s) relative to the uncompressed upload and 1.64 times relative to *gzip -6* (3.31 MB/s). For LOG.csv, *xz -0* improves the effective throughput 1.19 times (4.83 MB/s) relative to *gzip -6* (4.05 MB/s). For 5 MB/s network, the uncompressed uploads achieve 4.45 MB/s for BRW.csv and 3.16 MB/s for LOG.csv. For BRW.csv, *pigz -6* achieves 4.05-fold

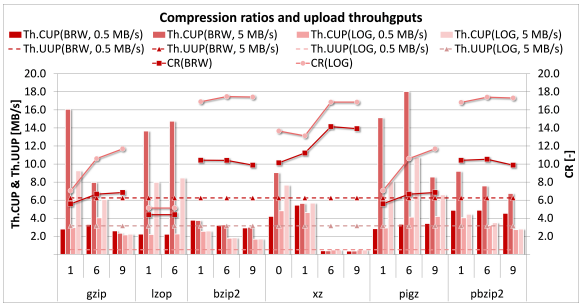

Fig. 2. Compression ratios and throughputs for uploads of mHealth files.

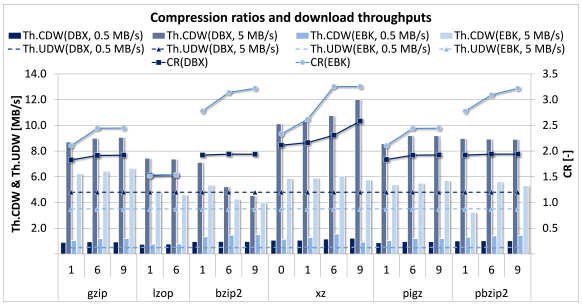

Fig. 3. Compression ratios and throughputs for downloads of applications and books.

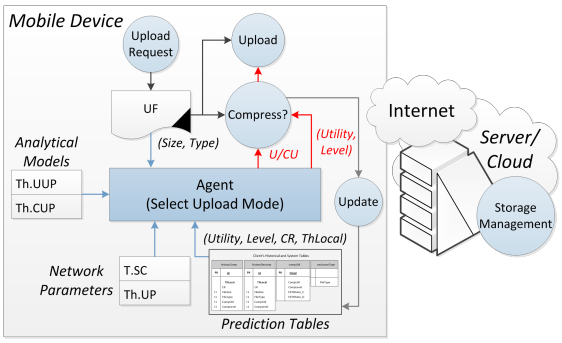

Fig. 4. System view of optimized file uploads.

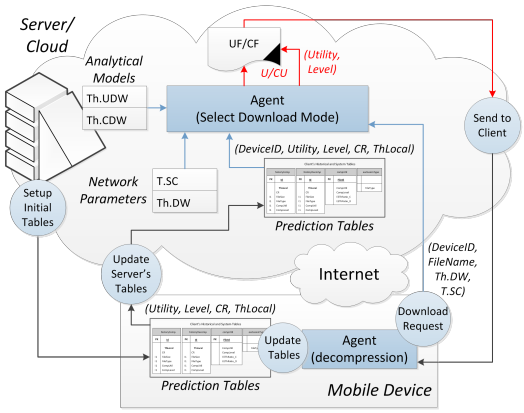

Fig. 5. System view of optimized file downloads.

improvement (18.03 MB/s) relative to the uncompressed upload and a 2.27-fold improvement relative to *gzip -6* (7.96 MB/s). For LOG.csv, *pigz -6* achives 1.77-fold improvement relative to *gzip -6*.

Download examples. An executable (DBX.tar - 69.3 MB) and an ebook (EBK.txt - 5.4 MB) are downloaded using different transfer modes from the cloud. All compressed versions of files are made available in the cloud. Fig. 3 shows compression ratio (*CR*), the uncompressed download throughput (*Th.UDW*), and the compressed download throughput (*Th.CDW*) for downloads over 0.5 MB/s and 5 MB/s networks. For downloads of DBX.tar over 0.5 MB/s network, *xz -9* achieves the best effective throughput, offering a 2.46-fold improvement (1.23 MB/s) relative to the uncompressed download and a 1.3-fold improvement relative to *gzip -6* (0.95 MB/s). For EBK.txt, *xz -6* offers 1.3-fold improvement relative to *gzip -6*. For downloads 5 MB/s network, the uncompressed downloads achieve the effective throughputs of 4.8 MB/s for DBX.tar and 3.5 MB/s for EBK.txt. For DBX.tar, *xz -9* achieves a 1.34-fold improvement over *gzip -6*. For EBK.txt, *gzip -9* slightly outperforms *gzip -6*.

These examples illustrate effectiveness of compressed file transfers as well as how file type, file size, level of redundancy, and network conditions impact the choice of best performing file transfer mode. Ideally, we would like to have on-demand agents that can autonomously, in real-time, with no significant overhead make a selection of a near optimal file transfer mode.

III. Proposed Framework

A. System View of Optimized File Transfers

Fig. 4 illustrates a system view of optimized file uploads initiated on a mobile device. An upload agent running on the device selects an effective transfer mode. To do so, it relies on the following: (i) file information (type, size); (ii) network connection parameters; (iii) analytical models describing the effective throughput for uncompressed as well as for compressed file uploads for all (utility, level) combinations supported on the device; and (iv) history-based prediction tables that *predict* the compression ratio and the local compression throughput for a given file on a given mobile platform.

The upload agent operates as follows. On a new upload request, the agent performs a query on the prediction table with the file size and type as inputs. The query produces estimated compression ratios and local compression throughputs for each (utility, level) pair supported on the device. The agent uses these estimates and the current network parameters in the analytical models to calculate the effective compressed and uncompressed upload throughputs. The estimated effective throughput of the best performing pair is compared to the estimated effective throughput of the uncompressed upload. If it

offers a higher throughput, the agent initiates the selected compressed upload. Otherwise, the file is transferred uncompressed.

Fig. 5 illustrates a system view of optimized file downloads initiated from a mobile device. A download agent running on the mobile device initiates a download request by sending the device *id*, file name, and the current network parameters to the cloud. Another agent running on the cloud maintains prediction tables as well as analytical models describing effective download throughputs for each mobile device.

The cloud agent operates as follows. When a new request is received, it performs a query on the prediction tables with the device *id* and the uncompressed file size and type as inputs. The query produces estimated compression ratios and the local decompression throughputs for each (utility, level) pair supported on the requesting device. Note: if the cloud already maintains compressed versions of the requested file, the actual compression ratios are known and do not have to be predicted. These estimates are then used in the analytical models to calculate the effective compressed download throughputs for each (utility, level) pair. The throughput of the best performing pair is compared to the effective throughput of the uncompressed download. If it offers a higher throughput, a file compressed with winning (utility, level) pair is sent from the cloud to the mobile device and the agent on the mobile device then initiates a decompression task. Otherwise, the uncompressed file is sent by the cloud to the mobile device.

The expected savings due to compressed file uploads or downloads should exceed by far the time the agent(s) spends in the selection process. To avoid imposing an additional overhead to the current file transfers, we assume that network parameters such as communication channel upload throughput, *Th.UP*, download throughput, *Th.DW*, and the connection setup time, *T.SC*, are all acquired prior to the actual transfer e.g., by periodically probing the communication channel. Once the transfer has been completed, the upload and download agents on the mobile device determine the compression ratio and the local (de)compression throughput and create a new entry in the prediction tables to inform future queries.

B. Analytical Models and Prediction Tables

The proposed framework relies on analytical models that estimate the effective upload [download] throughput of uncompressed and compressed file transfers. The total time for an uncompressed file transfer includes the time to establish the connection, *T.SC*, and the file transmission time, which depends on the files size (*US*) and the network throughput (*Th.UP [Th.DW]*). The effective upload throughput, *Th.UUP*, is calculated as the uncompressed file size in megabytes divided by the total upload time and can be calculated as shown in Eq. (1). Similarly, the effective download throughput, *Th.UDW*, is calculated as shown in Eq. (2). The expressions imply that the effective uncompressed throughputs reach the network throughputs when transferring very large files. In the case of small files, the time to establish the network connection significantly limits effective throughputs.

$$Th.UUP = \frac{Th.UP}{1+Th.UP\cdot\frac{T.SC}{US}} \quad (1)$$

$$Th.UDW = \frac{Th.DW}{1+Th.DW\cdot\frac{T.SC}{US}} \quad (2)$$

$$Th.CUP = \frac{CR\cdot Th.UP}{k.up+CR\cdot Th.UP\cdot(\frac{k.up\cdot T.SC}{US}+\frac{k.c}{Th.C})} \quad (3)$$

$$Th.CDW = \frac{CR\cdot Th.DW}{k.dw+CR\cdot Th.DW\cdot(\frac{k.dw\cdot T.SC}{US}+\frac{k.d}{Th.C})} \quad (4)$$

To extract unknown network parameters, *T.SC* and *Th.UP [Th.DW]*, we perform a two file upload [download] test. Two files of different sizes are transferred over a network connection that we want to characterize. The total times are measured and then used to calculate the effective network throughputs described in Eqs. (1) and (2). By replacing the measured *Th.UUP* and *US* for both files in Eq. (1), we get two equations with two unknowns, *T.SC* and *Th.UP*. Similarly, by replacing the measured *Th.UDW* and *US* in (2) for both files, we can determine *T.SC* and *T.DW* that characterize the download channel [12].

For compressed uploads and downloads we assume piped data transfers where local (de)compression tasks are partially or fully overlapped with file transfers. Eqs. (3) and (4) are used to estimate compressed upload and download throughputs. We start from an optimistic assumption where local compression and decompression tasks are fully overlapped with file transfers, and then introduce corrective factors (*k.up [k.dw]* and *k.c [k.d]*) to capture actual behavior. The degree of overlap depends either on the ratio between the network throughput and the maximum effective throughput (for *k.up [k.dw]*) or on the ratio between the network throughput and the local (de)compression throughput (for *k.c [k.d]*). A detailed description of presented throughput and similarly derived energy efficiency models can be found in [12].

The analytical models for estimating throughputs of compressed data transfers rely on predicting local throughputs on a mobile device, *Th.C [Th.D]*, and compression ratio, *CR*. The prediction relies on history logs captured on the device for each combination of compression utility and level. They are initially created with a set of representative data and updated with each new file transfer. The logs are stored in tables; each entry contains the following parameters: the compression utility and level, the uncompressed file size, *US*, the compression ratio, *CR*, and the (de)compression throughput, *Th.C [Th.D]*. The prediction relies on statistical redundancy in categories of files such as sensor-generated (physiological data, raw images), computer generated (code, binary), and human-readable data (books, documents, emails).

The data tables maintained by the mobile device agent are *historyComp*, *historyDecomp*, *compUtil*, and *exclusionType*. The *historyComp* table contains history logs for local compressions and includes the following fields: *US* and *Type* to describe the uncompressed file size and type, *C_Util* and *C_Level* to describe the corresponding compression (utility, level) pair (e.g., *"gzip"*, *"1"*), and *Th_C* and *CR* to describe the local compression throughput and compression ratio. The *historyDecomp* table similarly maintains history logs for local decompressions. Finally, the *compUtil* table maintains a list of all compression pairs available on the device. The *exclusionType* table maintains a list of uncompressible file types which are ignored and transferred uncompressed.

The cloud agent maintains tables similar to the tables on the mobile device. However, since cloud agent needs to maintain history logs and device characteristics for multiple mobile devices, additional tables are needed to perform long-term storage and initialization of mobile data tables. In addition, the *historyComp*, *historyDecomp*, and *compUtil* tables include the *Device_ID* field. The *deviceType* table includes a list of all supported devices, while the *fileStorage* table maintains remote and local locations of files.

C. Agent Implementation

The framework's mobile device agent is implemented in C++ with a SQLite database, selected due to its default Android support. The mobile agent is compiled for Android using NDK toolset. The implementation of the cloud agent utilizes MySQL to allow concurrent requests from the numerous devices.

Optimized uploads. Assuming the network parameters are set in advance and file information is retrieved from the file system, the mobile device performing optimized upload invokes its framework agent with a set of above parameters as arguments. With those parameters, the agent executes a set of SQL statements to query historyComp prediction table, uses analytical model to estimate the throughputs of compressed upload, sorts the query results, and finally performs a system call to invoke final selection for optimized transfer.

Fig. 6 shows the SQL statements generated for upload of 101.4 MB file over a WLAN interface with the network throughput of 5 MB/s and the connection setup time of 0.39 seconds. Lines 1-22 query the closest local compression throughput, *Th.C*, and compression ratio, *CR*, and create a temporary table with values for each (utility, level) pair. Lines 22-29 sort the temporary table using analytical model, select the result with the highest estimated throughput (e.g. *"pigz"*, *"1"*), and drop temporary table. After SQL execution, estimated effective throughput of selected (utility, level) pair is compared to the uncompressed upload throughput. If a compressed upload is selected, the compressed size, *CS*, and total execution time, *T.CUP*, are returned at the end of the upload process. The agent will then update prediction tables with compression ratio, *CR*, and estimated local compression throughput, *Th.C*. SQL execution being the critical component, the query is optimized to execute on a 30,000 entry table in 27.4 ms for upload.

```
1  // inUS=101.4; inType='csv'; inThUP=5.0; inTSC=0.362;
2  CREATE TEMPORARY TABLE tempt AS
3  SELECT T.C_Util,T.C_Level,Ext,CR,Th_C FROM (
4  SELECT C_Util,C_Level,(SELECT CR FROM (SELECT CR,diff FROM (
5    SELECT CR, abs(US-inUS) AS diff FROM historyComp AS T2
6    WHERE Type=inType and T2.C_Util=Tools.C_Util and
7    T2.C_Level=Tools.C_Level and US<=inUS
8    ORDER BY US DESC LIMIT 1)UNION ALL
9   SELECT CR,diff FROM (
10   SELECT CR,abs(US-inUS) AS diff FROM historyComp AS T2
11   WHERE US=inType and T2.C_Util=Tools.C_Util and
12   T2.C_Level=Tools.C_Level and US>inUS ORDER BY US DESC LIMIT 1) ORDER
          BY diff LIMIT 1)
13 ) AS CR,(SELECT Th_C FROM (SELECT Th_C, diff FROM (
14   SELECT Th_C,abs(US-inUS) AS diff FROM historyComp AS T3
15   WHERE Type=inType and T3.C_Util=Tools.C_Util and
16   T3.C_level=Tools.C_Level and US<= inUS
17   ORDER BY US DESC LIMIT 1) UNION ALL
18  SELECT Th_C,diff FROM (
19   SELECT Th_C,abs(US-inUS) AS diff FROM historyComp AS T3
20   WHERE Type=inType and T3.C_Util=Tools.C_Util and
21   T3.C_Level=Tools.C_Level and US>inUS ORDER BY US DESC LIMIT 1) ORDER
          BY diff LIMIT 1)) AS Th_C FROM compUtil AS Tools) AS T
22 LEFT JOIN compUtilExt ON (compUtilExt.C_Util=T.C_Util);
23 // Sort and select optimal pair
24 SELECT C_Util,C_Level,Ext,CR,Th_C FROM tempT
25 ORDER BY CASE WHEN Th_C>(1/(1/(CR*inThUP)+inTSC/US)) THEN
26 (1/((1/(CR*inThUP)+inTSC/US)+(inThUP/Th_C)/Th_C)) ELSE
27 (1/((1/CR+inThUP*inTSC/US)*(1/(CR*inThUP)+inTSC/US)+1/Th_C))
28 END DESC LIMIT 1;
29 DROP TABLE tempt;
```

Fig. 6. SQL query for uploading 101.4 MB file on 5 MB/s WLAN.

Optimized downloads. To perform optimized download, the mobile device sends download request to the cloud agent with input parameters set to include the network parameters, file information, and mobile devices id (e.g., A0001 for OnePlus One). Upon receiving the request, the cloud agent retrieves file size and type from its file system or from *fileStorage* table and executes a similar set of SQL statements on *historyDecomp* table to select a transfer mode with the highest effective throughput for the given device type. Selected transfer modes send either the compressed versions of files maintained on the cloud or perform on-demand compression of maintained uncompressed files. With 30,000 table entries, SQL query executes in 2.5 ms for download.

Once the mobile device starts receiving the file, it decodes the first incoming bytes to decide which decompression utility it should use to complete the transaction. If decompression is selected, the compressed size, *CS*, and total execution time, *T.CDW*, are returned at the end of the download process. The mobile device agent will then update its prediction tables with compression ratio, *CR*, and estimated local decompression throughput, *Th.D*. Updates to *historyDecomp* table will refine prediction data with each successful compressed download, and will be used periodically to update clouds tables.

IV. Experimental Evaluation

Experimental evaluation involves running upload and download agents on a mobile device and the cloud. Each transfer mode selected by the proposed framework is

TABLE II
DATASETS TO CHARACTERIZE LOCATION COMPRESSION

ID	Dataset Name	Type	# of Files	Size GB	Notes
D0	APK	apk	279	7.4	Extracted apk files
D1	apksource	code	59	1.2	Android source files
D2	Books	text	1067	0.6	Project Gutenberg ebooks
D3	DNG	dng	67	1.5	Lossless DNG images
D4	HealthSUM	csv/dat	28	0.07	Physiological log records
D5	HealthWAVE	csv/dat	86	2.9	Physiological waveforms
D6	Maps	tar	51	2.6	MAPS.ME offline maps
D7	Maps.route	tar	50	2.5	MAPS.ME offline routes
D8	OsmAnd	tar	50	7.6	OsmAnd navigation files
D9	Translate	tar	50	9.4	Google translation files

augmented to measure the effective upload and download throughputs. To measure the effectiveness of the proposed framework, the effective throughputs achieved by the framework, *Th.FW*, are compared to the effective throughputs of uncompressed transfers, *Th.UUP [Th.UDW]*, and the default compressed transfers that utilize *gzip -6*, *Th.CUP(gzip -6) [Th.CDW(gzip -6)]*. The measurements are repeated for WLAN network throughputs set to 0.5, 2, 3.5, and 5 MB/s

Datasets. To evaluate the effectiveness of the framework for various data files, a collection of representative datasets is compiled. The datasets include executables and source files of popular Android applications, text files, lossless images in the DNG format, mobile health summary and waveform files in text and binary formats, offline maps and routing files from MAPS.ME and OsmAnd applications, and offline language packages from the Google Translate application. Table II gives a complete list of the datasets used, including file types, the number of files in a set, the total size, a description, and dataset identifiers (*D0-D9*) used in the rest of this paper. Files that are compressed by default (e.g., *apk*) are repackaged into uncompressed archives (tar). The datasets are used in creating prediction tables for supported local compression utilities and levels. Finally, the number of entries in the tables is reduced and averaged to a certain file size granularity.

V. RESULTS

A. Throughput Speedup for Uploads

The first throughput speedup for uploads is calculated as the throughput achieved by the framework divided by the throughput of the corresponding uncompressed upload, *Th.FW/Th.UUP*. It varies as a function of file type, file size, and network parameters. The optimized file uploads are highly beneficial for the majority of files, except for a small group of book files. The maximum speedup ranges from 1.29 for *D7* files to 14.74 for the *D5* files on a 0.5 MB/s WLAN, and from 2.34 for the *D4* files to 6.71 for the *D5* files on the 5 MB/s WLAN.

The second speedup is calculated as the throughput achieved by the framework divided by the throughput of the default compressed upload, *Th.FW/Th.CUP(gzip -6)*. In the case of low network throughput (0.5 MB/s), the advantage of the optimized file uploads is highly depended on the file type. The maximum speedup ranges from 1.2 to 1.3 for *D0-D3*, from 1.15 to 1.6 for *D4-D5*, and effectively achieving no speedup for *D6-D9*. The framework selects utilities with a high compression ratio for easily compressible files (e.g., 91% *bzip2* for *D5.dat*, and 42% *xz* for *D1*) and faster utilities for files with limited compressibility (100% *pigz* for *D6-D9*). For high network throughput (5 MB/s), the optimized uploads provide significant improvements over the default compressed uploads. The maximum speedup ranges from 2.23 to 5.28 for *D0-D2*, from 2.85 to 11.56 for *D3*, from 4.78 to 11.32 for *D4-D5*, and from 3.44 to 5.58 for *D6-D9*. The framework selects *pigz* (50%-100%), *lzop* (up to 60% for *D2* and *D4*), and uncompressed transfer.

Table III shows the total upload throughput speedups for each and for all datasets combined (row Total) when transferring files over the WLAN network connection with the network throughputs set to 0.5, 2, 3.5, and 5 MB/s. To determine the total throughput speedup achieved by the framework for a dataset, we divide the sum of all total uncompressed file transfer times in the dataset with the sum of all total compressed file transfer times when using the framework. The total throughput speedup relative to uncompressed file uploads for all datasets used in the evaluation ranges from 1.46 (on the 3.5 MB/s network) to 2.38 (on the 5 MB/s network). The speedup increases with an increase in the network throughput. It should be noted that the optimized uploads are most beneficial for files that are most likely to be uploaded to the cloud, including the mHealth files and DNG images. The optimized transfers perform well for all network conditions. The total throughput speedup relative to the default compressed file uploads for all files used in the evaluation ranges from 1.02 (on the 0.5 MB/s network) to 2.97 (on the 5 MB/s network).

B. Throughput Speedup for Downloads

The first speedup for downloads is calculated as the throughput achieved by the framework divided by the throughput of the corresponding uncompressed download. The optimized file downloads are highly beneficial for the majority of files, except for a small group of book files transferred over a high-throughput connection. The maximum speedup ranges from 1.37 for the *D7* files to 32.49 for the *D5* files on the 0.5 MB/s network, and from 2.77 for the *D7* files to 18.28 for *D5* files on the 5 MB/s network.

The second speedup is calculated as the throughput achieved by the framework divided by the throughput of the default compressed download, *Th.FW/Th.CDW(gzip -6)*. For low network throughput, the maximum speedup ranges from 1.20 to 1.78 for *D0-D3*, from 1.51 to 3.8 for *D4-D5*, and from 1.08 to 1.25 for *D6-D9*. The framework selects *bzip2* (91% for *D2*), *xz* (86%-100%), and *pbzip2*

TABLE III
UPLOAD SPEEDUPS: TH.FW/TH.UUP [TH.CUP(GZIP)]

	0.5 MB/s		2 MB/s		3.5 MB/s		5 MB/s	
Th.FW/Th.	**UUP**	**gzip**	**UUP**	**gzip**	**UUP**	**gzip**	**UUP**	**gzip**
D0	2.29	1.05	2.12	1.38	1.86	1.97	1.84	2.70
D1	3.25	1.03	2.82	1.21	2.72	2.14	2.95	2.30
D2	1.95	1.01	1.27	1.10	1.09	1.20	1.10	1.14
D3	2.40	1.28	1.84	2.91	1.74	4.55	2.52	6.40
D4.csv	7.24	1.16	3.20	1.33	2.53	1.76	2.17	1.80
D4.dat	5.42	1.15	2.68	1.52	1.96	1.52	1.83	1.44
D5.csv	12.28	1.57	7.17	3.27	5.97	4.43	5.35	5.83
D5.dat	4.51	1.37	2.79	2.55	2.18	3.60	2.75	4.04
D6	1.26	1.00	1.24	1.06	1.23	1.61	2.23	2.41
D7	1.14	1.00	1.13	1.26	1.14	1.93	2.03	2.71
D8	1.35	1.00	1.33	1.24	1.34	2.14	2.64	3.24
D9	1.28	1.00	1.27	1.26	1.28	1.84	2.52	2.80
Total	1.56	1.02	1.50	1.32	1.46	2.06	2.38	2.97

TABLE IV
DOWNLOAD SPEEDUPS: TH.FW/TH.UDW [TH.CDW(GZIP)]

	0.5 MB/s		2 MB/s		3.5 MB/s		5 MB/s	
Th.FW/Th.	**UDW**	**gzip**	**UDW**	**gzip**	**UDW**	**gzip**	**UDW**	**gzip**
D0	2.72	1.26	2.87	1.22	2.44	1.15	2.06	1.01
D1	4.12	1.31	3.93	1.29	3.21	1.14	3.64	1.00
D2	1.92	1.17	1.21	1.25	1.15	1.07	1.12	1.24
D3	2.78	1.47	2.49	1.49	2.28	1.27	3.16	1.13
D4.csv	7.11	1.21	3.62	1.08	2.61	1.24	2.30	1.04
D4.dat	5.72	1.29	2.84	1.13	2.10	1.07	1.83	1.02
D5.csv	22.18	2.84	15.68	2.13	7.88	1.14	11.8	1.25
D5.dat	4.58	1.40	3.15	1.15	2.50	1.05	3.08	1.04
D6	1.45	1.15	1.44	1.15	1.25	0.99	2.30	1.02
D7	1.21	1.06	1.15	1.01	1.15	1.00	2.12	1.01
D8	1.63	1.20	1.60	1.18	1.36	1.00	2.51	0.98
D9	1.55	1.20	1.53	1.19	1.29	1.00	2.55	1.04
Total	1.85	1.20	1.78	1.19	1.53	1.02	2.50	1.03

(100% for *D3* and 60% for *D5.dat*). For higher network throughput, the advantage of the optimized file downloads is highly depended on the file type. The maximum speedup ranges from 1.01 to 1.40 for *D0-D3*, from 1.18 to 1.7 for *D4-D5*, and from 1.00 to 1.09 for *D6-D9*. The framework selects fast utilities such as *pigz* (31% for *D1* to 100% for *D8-D9*), *lzop* (31% for *D2*), and *gzip*, as well as slower utilities such as *xz* (43% for *D4.csv*) and *pbzip2* (100% for *D3* and *D5.csv*).

Table IV shows the total download throughput speedup for each and for all datasets combined (row Total) when transferring files over the WLAN network with the network throughput set to 0.5, 2, 3.5, and 5 MB/s. The total throughput speedup relative to uncompressed file downloads for all files used in the evaluation ranges from 1.53 (on the 3.5 MB/s network) to 2.5 (on the 5 MB/s network). The total throughput speedup relative to the default compressed file downloads for all files used in the evaluation ranges from 1.02 (on the 3.5 MB/s network) to 1.2 (on the 0.5 MB/s network).

VI. CONCLUSION

This paper describes the design and an implementation of the framework for optimizing data file transfers between mobile devices and the cloud using compression utilities. Agents running on mobile devices and the cloud utilize file information, network connection parameters, analytical models for describing effective throughput of uncompressed and compressed file transfers, and history-based tables for predicting compression ratio and local (de)compression throughputs to select a transfer mode that maximizes the effective file transfer throughput. The effectiveness of the proposed framework is experimentally evaluated for a range of files by comparing the effective throughput of optimized file transfers to the effective throughput of uncompressed file transfers and file transfers that use the default compression. We showed that the proposed framework significantly improves the effective throughput and outperforms the default compressed transfers for certain network conditions.

REFERENCES

[1] IDC, "Apple, Huawei, and Xiaomi Finish 2015 with Above Average Year-Over-Year Growth, as Worldwide Smartphone Shipments Surpass 1.4 Billion for the Year, According to IDC," 2016. [Online]. Available: http://tinyurl.com/hn62p75

[2] Cisco, "Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update, 20152020 White Paper," 2016. [Online]. Available: http://tinyurl.com/gupkde5

[3] V. Agababov, M. Buettner, V. Chudnovsky, M. Cogan, B. Greenstein, S. McDaniel, M. Piatek, C. Scott, M. Welsh, and B. Yin, "Flywheel: Google's Data Compression Proxy for the Mobile Web," in *Proceedings of the 12th USENIX Conference on Networked Systems Design and Implementation*, ser. NSDI'15. Berkeley, CA, USA: USENIX, 2015, pp. 367–380.

[4] K. C. Barr and K. Asanović, "Energy-aware lossless data compression," *ACM Transactions on Computer Systems*, vol. 24, no. 3, pp. 250–291, Aug. 2006.

[5] A. Milenković, A. Dzhagaryan, and M. Burtscher, "Performance and Energy Consumption of Lossless Compression/Decompression Utilities on Mobile Computing Platforms," in *2013 IEEE 21st International Symposium on Modeling, Analysis Simulation of Computer and Telecommunication Systems (MASCOTS)*, San Francisco, CA, Aug. 2013, pp. 254–263.

[6] A. Dzhagaryan, A. Milenković, and M. Burtscher, "Quantifying Benefits of Lossless Compression Utilities on Modern Smartphones," in *2015 24th International Conference on Computer Communication and Networks (ICCCN)*, Las Vegas, NV, Aug. 2015, pp. 1–9.

[7] A. Dzhagaryan and A. Milenković, "On Effectiveness of Lossless Compression in Transferring mHealth Data Files," in *2015 17th International Conference on e-Health Networking, Applications and Services (Healthcom)*, Boston, MA, Oct. 2015, pp. 665–668.

[8] C. Krintz and S. Sucu, "Adaptive on-the-fly compression," *IEEE Transactions on Parallel and Distributed Systems*, vol. 17, no. 1, pp. 15–24, Jan. 2006.

[9] B. Nicolae, "High Throughput Data-Compression for Cloud Storage," in *Data Management in Grid and Peer-to-Peer Systems*, ser. Lecture Notes in Computer Science, A. Hameurlain, F. Morvan, and A. M. Tjoa, Eds. Springer Berlin Heidelberg, Sep. 2010, no. 6265, pp. 1–12.

[10] D. Harnik, R. Kat, O. Margalit, D. Sotnikov, and A. Traeger, "To Zip or Not to Zip: Effective Resource Usage for Real-time Compression," in *Proceedings of the 11th USENIX Conference on File and Storage Technologies*, ser. FAST'13. San Jose, CA: USENIX, 2013, pp. 229–241.

[11] A. Dzhagaryan, A. Milenković, M. Milosevic, and E. Jovanov, "An environment for automated measurement of energy consumed by mobile and embedded computing devices," *Measurement*, vol. 94, pp. 103–118, Dec. 2016.

[12] A. Dzhagaryan and A. Milenković, "Models for Evaluating Effective Throughputs for File Transfers in Mobile Computing," in *2016 25th International Conference on Computer Communication and Networks (ICCCN)*, Waikoloa, HI, Aug. 2016, pp. 1–9.