The Cloudlet Accelerator: Bringing Mobile-Cloud Face Recognition into Real-Time

Nathaniel Powers*, Alexander Alling*, Kiara Osolinsky*, Tolga Soyata*,
Meng Zhu*, Haoliang Wang†, He Ba*, Wendi Heinzelman*, Jiye Shi‡, Minseok Kwon§

*Department of ECE
U. of Rochester, Rochester, NY, 14627
{npowers,aalling,kosolins,soyata,hba,
mzhu13,wheinzel}@ece.rochester.edu

†Department of CS
George Mason University
Fairfax, VA 22030
dut.wong@gmail.com

‡Computational Biology
UCB Pharma
Rochester, NY 14623
jiye.shi@ucb.com

§Department of CS
Rochester Inst. of Tech.
Rochester, NY 14623
mjk@rit.edu

***Abstract*—A mobile-cloud architecture provides a practical platform for performing face recognition on a mobile device. However, using a mobile-cloud architecture to perform *real-time* face recognition presents several challenges including resource limitations and long network delays. In this paper, we determine three approaches for accelerating the execution of the face recognition application by utilizing an intermediate device called a *cloudlet*. We study in detail one of these approaches, using the cloudlet to perform *pre-processing*, and quantify the maximum attainable acceleration. Our experimental results show up to a $128\times$ improvement in response time when appropriate cloudlet hardware is used.**

I. Introduction

Object detection and recognition, in particular face recognition, have been given a great deal of attention because of its useful applications such as public security surveillance and automated robotics-based manufacturing. For applications like face recognition, instead of using webcams attached to PCs, it is more convenient and natural for users to use smartphones or tablets to detect and recognize faces. With the rapid development of semiconductor technology, more functions are now being integrated into mobile devices on which more sophisticated applications may run. However, even though mobile devices can do much more than before, their limited computational capabilities and battery life prevent their use in intensive augmented reality applications like real-time face detection and recognition in complex scenes. One solution to address these limitations is to utilize a cloud-based approach, which offloads the computationally-intensive parts of the application to cloud servers.

A mobile-cloud implementation of real-time face recognition has a significant challenge: the mobile device needs to send the entire image to the cloud server. Since a wide area network (WAN) connection is high latency and limited bandwidth (which is unlikely to improve in the near future [1]), sending the entire image dramatically increases the application response time, which is contrary to the goal of real-time or near-real-time face recognition [2]. Although one might try to decrease the communication cost by detecting faces on the mobile devices and sending only the faces to the cloud servers, our investigation shows that this approach will not work under complex and demanding situations, even with dedicated hardware for acceleration [3], [4].

In this paper, we introduce a mobile-cloudlet-cloud architecture where the cloudlet is a resource-rich device located nearby mobile devices. During the course of face recognition, intermediate computationally-intensive operations like face detection and projection [5] can be handed to the cloudlet over a low-latency local area network (LAN). The cloudlet can significantly reduce the amount of data transmitted over the high-latency WAN by performing pre-processing and data reduction methods. In the most extreme case (utilizing a minimal database), the post-projection data transmitted across the WAN for each face is less than 1% of raw image size.

Our contributions in this paper are summarized as follows: i) We evaluate the potential acceleration in performing face recognition on a mobile device by using a cloudlet, ii) We present our results using an actual implementation of a face recognition application running on an Android mobile phone, an x86-based cloudlet utilizing an Nvidia GPU, and a cloud server simulating network latencies based on measured data over Planet-Lab [6], [7]. iii) We study the *pre-processing* capability of the cloudlet and quantify the improvement in the application response time attainable due to pre-processing.

The rest of the paper is organized as follows. In Section II, we provide background information on different approaches for mobile-cloud computing and their limitations. In Section III, we provide pointers to how the cloudlet can improve the application response time. In Section IV, we introduce the details of the face detection and recognition algorithms we use, followed by a description of their implementation in our mobile-cloudlet-cloud system. Our experimental results are described in Section V, and our conclusions and pointers to future work are provided in Section VI.

II. Background

Processing capabilities of modern mobile devices have improved dramatically in recent years. However, there are significant constraints on smart devices in terms of size, weight, processing speed, memory, storage capacity, and battery life [4]. Computationally-demanding and storage-heavy mobile applications are still beyond the capabilities of today's smartphones and tablets. To extend the capabilities of smart devices, mobile-cloud computing was introduced that allows mobile applications to leverage cloud resources [8]–[14].

In general, there are two approaches to improving the performance of mobile-cloud applications [15]. One is computation offloading, that is, executing the computationally-complex components of the application at the resource-rich cloud. The other approach is to use a task distribution algorithms to distribute different parts of the application to different cloud resources in order to obtain the optimal performance in terms of battery life and application response time. Both of these approaches can be achieved by either 1) transmitting the computationally-complex code to the cloud and executing it there using Virtual Machine (VM) techniques [16], or 2) transmitting the data to a cloud that already has the code to process the data. Both approaches trade-off computational load for communications load on the mobile device.

VM-based techniques allow code to be executed on both the mobile platform and the cloud platform, providing the convenience of code migration [8]–[11], [17]. Developers may benefit from this in that they do not have to program on multiple platforms and therefore the time to develop a mobile-cloud application can be reduced. However, this implies a performance penalty for two reasons : 1) Executing code through a VM means an extra translation step from the actual code to the native code, 2) the transmission of the code along with data adds a significant amount of communication latency. On the other hand, applications with code specifically programmed for and installed on mobile devices, cloudlets and cloud servers have better performance but may lose the portability and code-development-time advantages.

Task distribution algorithms are helpful to fully utilize the cloud resources [1], [13], [14]. Especially when multiple cloud options exist, for example, Amazon Web Services (AWS) [18], Microsoft Windows Azure [19], Google Cloud Platform [20] or even mobile devices such as Hyrax [21] and GEM-Cloud [22], an algorithm is needed for the selection of which cloud to use based on the network conditions, processing speeds, storage, availabilities, etc. In order to apply a task distribution algorithm, a profiler is needed to estimate the resources required by both the mobile device and the cloud in terms of time and energy, and the network conditions in terms of bandwidth, packet loss and latency [7]. Algorithms can determine the optimal way to distribute tasks based on the information gathered by the profiler. Usually this involves solving an optimization problem whose objective functions are to maximize battery life or minimize application latency.

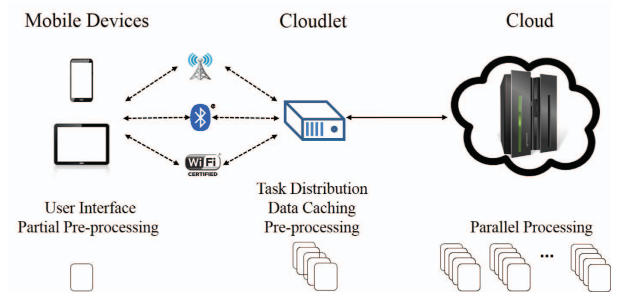

Fig. 1: Acceleration can be achieved by utilizing the cloudlet for pre-processing, intelligent task distribution and data reduction.

III. Utilizing a Cloudlet as an Accelerator

Although mobile devices have improved dramatically over the past few years, they are still relatively limited in processing speed, memory, storage, battery life and network bandwidth. For latency-sensitive or real-time applications [23]–[25], it is necessary to reduce the application response time to provide the best user experience [26]. Because of the inconsistent network conditions over the internet and the possible unavailability of cloud servers, a cloudlet can be introduced to provide local computing power and storage and the intelligence for task management. Figure 1 shows an example of a mobile-cloud architecture that utilizes a cloudlet as a local edge server that can communicate with the mobile over a LAN.

In order to reduce application response time, there are three ways that a cloudlet may help:

1) It can use its superior computing capability to pre-process and reduce the data that must be transmitted to the cloud, thereby reducing communication times.
2) It can utilize its rich storage capacity to cache a portion of a large database from the cloud, so that frequently-needed database elements can be delivered over the local network to the mobile device when it needs them.
3) It has the ability to serve multiple mobile devices and offload tasks to multiple cloud servers. The cloudlet can profile all of the available resources (i.e., cloud servers) in the network and then execute task-distribution algorithms in order to optimize overall performance.

In this paper, we introduce a cloud-based face recognition system that allows mobile devices to perform near-real-time face recognition by taking advantage of the resources made available by the cloudlet and the cloud. Cloudlet pre-processing and data reduction can reduce the communication time at the expense of computation [27]. If this computation time within the cloudlet can be minimized, there is a case for acceleration. Potentially, the cloudlet can cache a portion of the database that is permanently stored in the cloud, thereby reducing the bandwidth burden on the mobile-cloud link. Furthermore, the cloudlet is able to serve multiple mobile phones at the same time and is able to optimize their performance by applying a scheduling algorithm. In this paper, we will focus on the analysis of the benefits of using a cloudlet as a dedicated pre-processor for our face recognition application.

IV. The Mobile-Cloud Face Recognition System

In this section, we will focus on the computational requirements of Face Recognition (**FR**) and how we implement FR on our mobile-cloudlet-cloud system. FR application contains three steps: face detection (abbreviated **FD**), conversion of the detected face to a set of coefficients, forming an eigenvector (an operation defined as projection in the literature, **PJ**) and a database search (denoted **S**). Note that, PJ and S steps are highly interrelated. We will now detail each one of these steps.

A. Face Detection (FD)

FD step isolates the useful information (the face) from the entire raw image. Since FD is not database-dependent, it can be separated from the subsequent steps and performed on any device with adequate computational power [28]. This gives us an opportunity to accelerate FD using a cloudlet. We used the Viola-Jones object detection framework [29]–[31] for FR.

Modern mobile devices incorporate System-on-Chips (SoCs), such as TMS320DM36x from Texas Instruments [32] to detect faces without consuming much CPU time and energy. Additionally, modern mobile operating systems such as Android 4.0 Ice Cream Sandwich, have a built-in API for the camera module [33], and with the help from these SoCs, FD can be performed in real-time while previewing from the camera. However, legacy mobile phones do not have such an accelerator and even with such an acceleration, the maximum number of faces is limited (e.g., max. two faces per frame on the LG Nexus 4 with Android 4.2 Jelly Bean). FD function is mainly introduced to support camera focusing. Additionally, FD computational demand is directly related to the resolution of the image. A resolution of 1280x720 is typical for phones introduced after 2011. Distinguishing a face from a crowd of 100's (or more) faces requires higher resolution, and has a much higher computational demand.

Using the OpenCV library [34] and Intel Threading Building Blocks (TBB) [35], the multi-threaded CPU version of FD took around 1500 ms to detect 17 faces in a 1280x720 images, which already eliminates any opportunity for real-time FD. Sending the entire image to a remote server dos not help either, since it will place a burden on the network bandwidth and will result in an increased application response time. To exemplify, sending 1280x720 images (200KB) at 7 frames per second (fps) implies a data rate of 11Mbps, which is very demanding even for modern WAN, 3G, and 4G networks. Adding a cloudlet between the LAN and WAN can help us achieve the goal of real-time FD [15]. The cloudlet can be orders of magnitude more powerful than mobile devices in terms of computation by incorporating a powerful GPU. Figure 2 depicts FD execution times on a 800x480 image using an Nvidia GPU takes 20–40 ms.

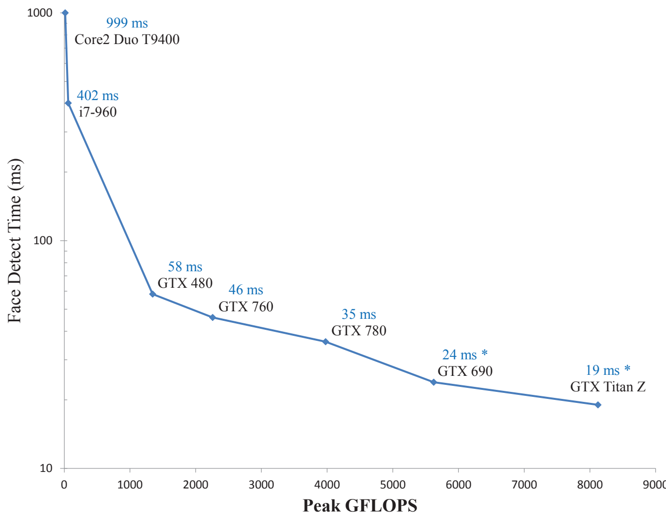

Fig. 2: FD execution times for 800x480 images (* means estimated).

B. Projection (PJ)

Although FD lets us separate the face from the background, the face image still contains redundant information for recognition. To further reduce the amount of information, we can use an eigenface approach [36], where, the detected face is projected onto the eigenface space to generate the eigenvector, which makes comparisons with potential faces much less computationally-intensive. The eigenfaces are generated from a database by using Principle Component Analysis (PCA) to extract a low-dimension representation from the higher-dimension image space (i.e., the database).

Thanks to the PCA, we can ignore a large portion of the eigenfaces without losing much precision by keeping those with the eigenvalues that are highest in magnitude. The eigenvalue associated with each eigenface represents how much an image in the database varies from the mean image in that direction. The eigenfaces with small eigenvalues only contain information about detailed differences, which is not critical under most cases. Figure 3 shows the relationship between the number of eigenfaces and the percentage of information of the entire database covered by the first k eigenfaces ($P(k)$), which is calculated as shown in Equation 1,

$$P(k) = \sum_{i=1}^{k} \left(\sum_{j=1}^{n} \Omega_{i,j} \right)^2 / \sum_{i=1}^{n} \left(\sum_{j=1}^{n} \Omega_{i,j} \right)^2 \qquad (1)$$

where $\Omega_{i,j}$ is the i^{th} component (eigenvalue) of the eigenvector for the j^{th} image in the database and n is the total number of images in the database. As can be seen in Figure 3, a small number of eigenfaces can cover most of the information in the database. Another interesting observation here is that the smaller a database size is, the higher ratio of eigenfaces is necessary to cover the same percentage of information. We used a heuristic principle for how to determine the number of eigenfaces : 8x smaller than the number of total images in the database which covers sufficient information for accurate FR.

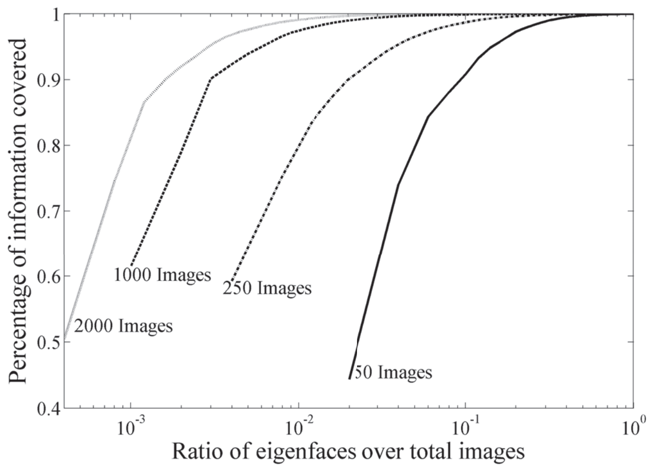

Fig. 3: % information covered with an increasing number of Eigenfaces.

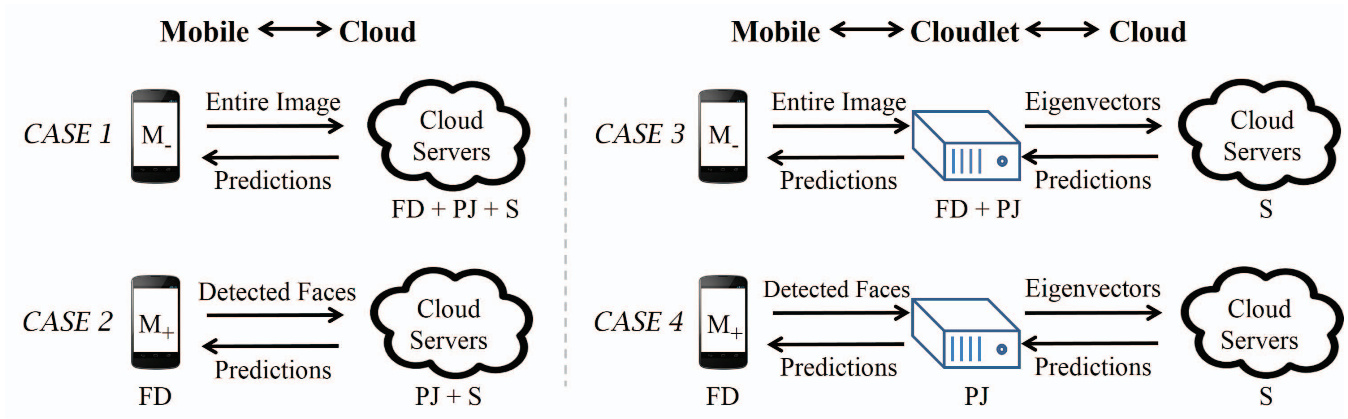

Fig. 4: Mobile-cloud FR using four options: Two with and two without using a cloudlet. M_+ and M_- notations are used to denote devices with and without built-in FD acceleration, respectively.

C. Database Search (S)

To recognize faces, we use neighbor search (NNS) and minimize the distance between the test face and the k^{th} face in the database $d_E(k)$ as shown in Equation 2,

$$d_E(k) = \sqrt{\sum_{i=1}^{n} \left(\Omega_{i,testface} - \Omega_{i,k}\right)^2} \qquad (2)$$

where Ω^i is the i^{th} component of the eigenvector for the k^{th} face in the database and *n* is the total number of eigenfaces. The face whose eigenvector yields the nearest distance will be the prediction of the input face. This is typically done by the cloud since the cloud generally has more storage space for the database than the cloudlet.

V. Performance Evaluation

A. Experimental Setup

Figure 4 illustrates four evaluated cases: The M_+ case was evaluated using an LG Nexus 4 smartphone with Android 4.2 Jelly Bean OS, which supports a built-in real-time FD API, though it is only capable of detecting up to two faces per frame. The mobile device compresses the entire acquired frame (in the M_- case), or up to two faces (in the M_+ case) to JPEG and sends them to the cloudlet for processing. For M_+ devices, the Android API makes its best efforts to acquire, compress and send every frame, but certain incoming frames might be skipped if the hardware acceleration limit has been reached. In our experiments, the rate was around 7 fps at a resolution of 1280x720.

The cloudlet has high-speed connections to the mobile device (e.g., 802.11n), and amassess two-orders-of-magnitude higher processing power. We emulated a cloudlet using two different Windows workstations. The first consisted of an Intel i7-960 Bloomfield CPU and an Nvidia GTX 480 GPU, and the second consisted of an Intel i7-4770K Haswell CPU and an Nvidia GTX 760 GPU. To isolate our system from the general public network traffic, we placed a dedicated wireless router between the mobile and the cloudlet.

We emulated a cloud server using a Windows workstation with an Intel i7-960 Bloomfield CPU with a Nvidia Tesla C2075 GPU and an Intel i7-4770K Haswell CPU with a variety of Nvidia GPUs: GTX 480, GTX 760 and GTX 780. In our system, the cloud server is capable of handling each data type and performing respective FR functions (FD, PJ and S) as required. By referencing the established performance baseline, it becomes clear that in order to achieve a tangible acceleration of the FR process with larger databases ($\rhd$2000 images), there is a necessity for significant computational power within the cloud. If the cloud server is indeed much more powerful than the cloudlet (and is not heavily loaded), the cloudlet may opt to offload a portion of its processing load to the cloud with the aim to reduce the overall response time.

The cloud server may receive 1) raw images from the mobile to perform FD, PJ and S, 2) detected faces from the mobile to perform PJ and S, 3a) projected vectors from the FD-enabled cloudlet to perform S, 3b) detected faces from the FD-enabled cloudlet to perform PJ and S and 4) eigenvectors from the FD-disabled cloudlet to perform S. Finally, the cloud sends the result back through the cloudlet to the originating mobile device. Although the connection between the cloudlet and the cloud server is LAN in our experiments, we added latencies to the response time on the cloud server to simulate real WAN conditions. The latency data was obtained through our experiments as described in our work [7]. Real latency data is obtained from experiments conducted on Planet-Lab where computers are physically located all over the world. Since mobile phones are typically served by cloud servers located on the same continent, we chose three servers in North America: Albuquerque, New Mexico; Ottawa, Ontario; and Santa Barbara, California. The databases we used were sourced from the MUCT face database [37], which contains 273 distinct individuals with 10 to 15 images each, and the University of Essex Face Recognition Data [38], which contains 395 individuals with 20 images each.

B. Experimental Results

Figure 5 depicts the results of our experiments with each Case depicted in Fig. 4. We are interested in the performance

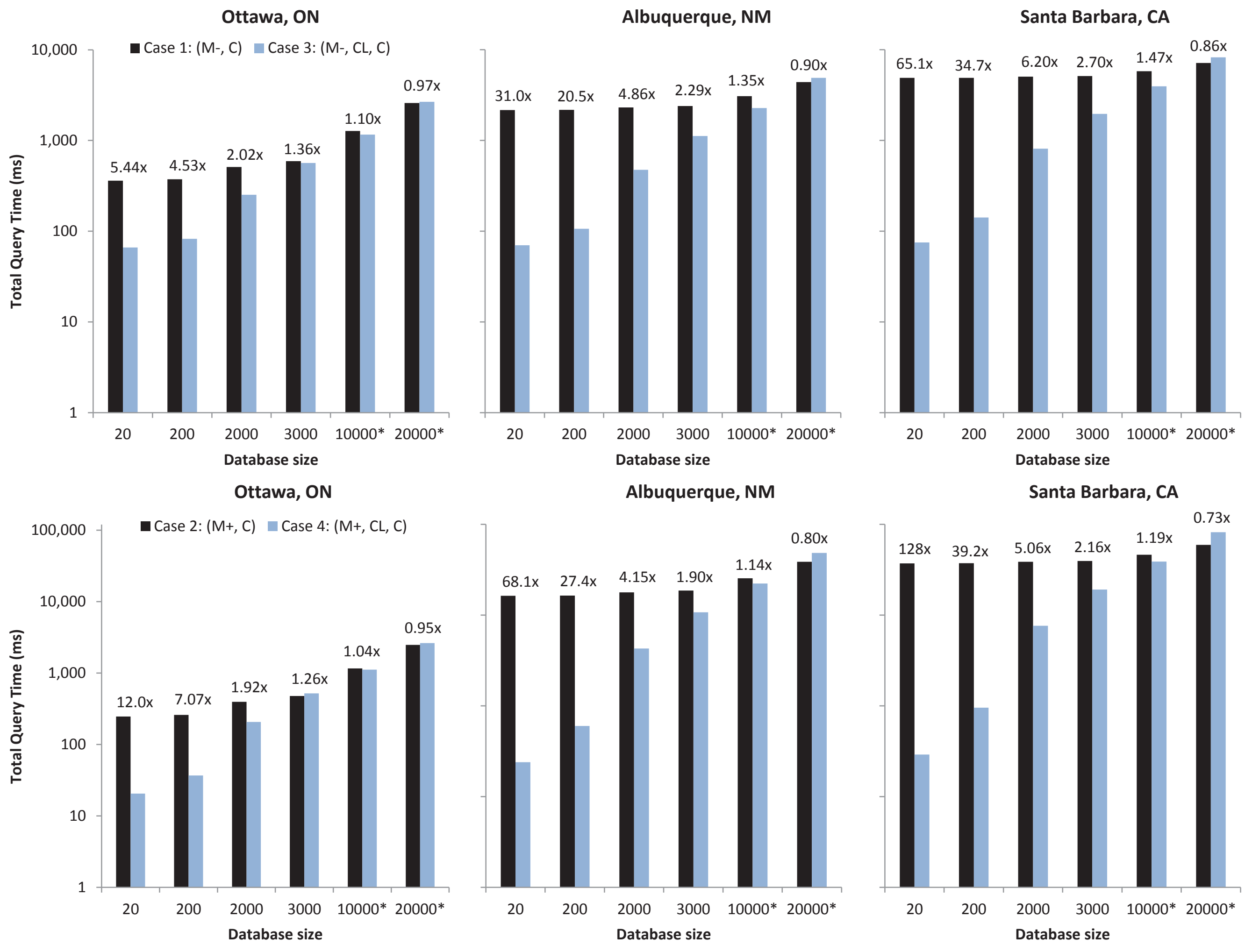

Fig. 5: Response time of the FR application with 800 x 480 image resolution and growing database-sizes for each case depicted in Fig. 4. Baseline GTX 760 platform used as cloudlet and cloud. Entries (marked *) above 3000 are extrapolated.

gain when we add a cloudlet to perform the pre-processing. As illustrated in Fig. 4, the baseline is when there is no cloudlet; the mobile sends the entire image or detected faces to the cloud as in Cases 1 and 2, respectively. By inserting the cloudlet into the architecture we see significant reductions in query times when the database is relatively small (nearly **128x**), as seen for Cases 3 and 4. The addition of the cloudlet vastly increases the upload speed of the mobile frames since they travel over LAN in lieu of WAN. The acceleration provided by the cloudlet, however, steadily decreases with database size because we have introduced a database-dependent vector type. This imposes a communication delay upon the WAN that eventually surpasses the delay reduction furnished by the LAN, resulting in a net *deceleration* of the communication component of the FR process. Assuming a vastly powerful cloud server, further data reduction methods can be employed by the cloudlet to mitigate such vector-database dependency.

PJ times for Haswell i7-4770K and various GPUs are depicted in Fig. 6. Cumulative moving averages were calculated over several minutes of continuous FR queries with various devices and database sizes. PJ times for each device increase linearly with database size, however the rate of increase for the GPUs show proportionality to the device clock speed, not the number of CUDA cores or the theoretical peak device GFLOPS as might be initially expected. This explains why the recorded PJ times for the GTX 760 (2258 GFLOPS, 980 MHz base clock) are much less than those of the GTX 780 (3977 GFLOPS, 863 MHz base clock).

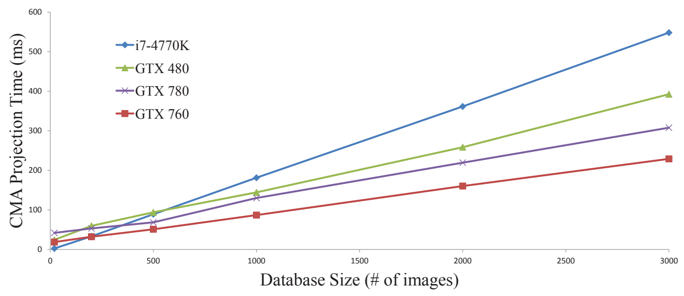

Fig. 6: Results of a cumulative moving average for PJ in different devices over several database sizes.

To carry our experimental results further, we formulated a device-compute-power relation for FD, PJ and S process

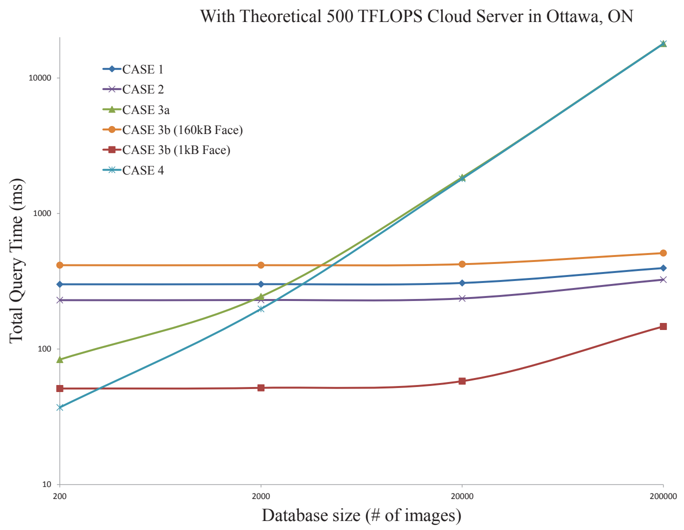

Fig. 7: Total Query Time (est.) of various FR schemes. PJ in the cloudlet (Case 3a and Case 4) shows significant database dependency. Query time dependency on face-data during PJ in the cloud (Case 3b) also illustrated for upper and lower size boundaries.

times, then extrapolated a cloud model located in Ottawa, ON with a performance of approximately 500 TFLOPS. The simulated query times for each of the four studied cases are represented in Fig. 7. Extending Case 3a is Case 3b, where the cloudlet is dedicated solely to performing FD and furnishing a LAN connection to the mobile device. As we have seen, cases involving PJ in the cloudlet (*Case 3a* and *Case 4*) produce minimized query times until the database reaches a certain size. Since we assume the cloud to have much greater compute-power than the cloudlet, its PJ time will not grow as quickly. Therefore, we may offload PJ to the cloud by opting to pass face data. Cloud PJ must occur below a threshold time interval defined by parameters entirely available to the cloudlet. We formalize the relationship in Eq. 3 below,

$$t_{PJcloud} < t_{PJcloudlet} - \left(\frac{face_{Bytes} - vector_{Bytes}}{WAN_{kBps}} \right) \quad (3)$$

Extrapolated FR performance boundaries for our recorded face data sizes are represented in Fig. 7 as *Case 3b, 160kB Face* and *Case 3b, 1kB Face*. Use of the face data type in lieu of the vector type reduces the database size dependency of the overall FR process. Because of this, for FR applications utilizing large databases, it may be beneficial to down-sample each face to a minimal size where PJ can always be handled by the cloud, with a low provisional impact on the FR result accuracy. We can schedule the cloudlet to perform PJ in other cases, such as momentary lag over the WAN.

Pivotal variables in Eq. 3 are the WAN speed and face data size. Since face data is proportional to its pixel-occupancy in the source frame, we have found that it could indeed be much smaller than the projected eigenvector. In Fig. 8, this case is represented by the *0.4kB Face* and the *16kB Face* curves. We would prefer sending the face over the vector because the *negative* differential (face $<$ vector) gives the cloud processing headroom, meaning, it may take longer to perform PJ than the

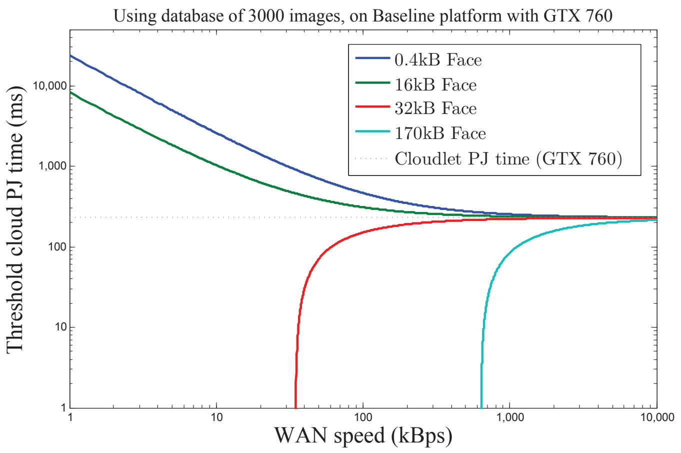

Fig. 8: Theoretical maximum cloud projection time resulting in reduced total query time, with dependency on WAN speed and face data size.

cloudlet would, but still net a reduced query time. Illustrated by the *32kB Face* and *170kB Face* curves, as WAN speeds decrease, the growing *positive* differential (face $>$ vector) between communicating a face or a vector across the WAN leads to a cut-off point where the cloud must be expected to perform PJ in zero time or less. Below this point, PJ in the cloudlet is the best solution.

This issue introduces the need for intelligence within the cloudlet in order to efficiently distribute tasks. The cloudlet has the capacity to be aware of parameters such as cloud and cloudlet performance, respective database sizes and real-time WAN speed (by periodically pinging the cloud on a defined frame interval) which contribute to an estimate for a threshold face data size. We have also embedded the cloud PJ time within each result data packet (when applicable). The cloudlet is able to skim this value from the result before passing it on to the mobile device, thereby forming a cumulative moving average for PJ performance in the cloud. To inform the decision to offload projection to the cloud, Eq. 3 is evaluated in the cloudlet with respect to each active cloudlet-cloud pair, so an optimal solution could be chosen *in real-time*.

VI. Conclusions and Future Work

We presented a mobile-cloudlet-cloud architecture to perform real-time face recognition by executing this application in three distinct steps: Face Detection (FD), Projection (PJ) and Searching (S). We observed that, due to their separability, these three steps can be executed in different hardware components: Mobile device (M), CLoudlet (CL), and Cloud (C). While M and CL components are connected through a single-hop communication link (i.e., 802.11n or 802.11ac LAN), allowing large chunks of data to be transferred in a small amount of time, the link between CL and C is much slower due to the multi-hop internet connection. We provided a detailed study of utilizing a cloudlet to accelerate the three aforementioned operations and determined that the FD and PJ operations can both benefit from a hardware-based acceleration.

We provided a detailed analysis of the computational requirements of the FD and PJ operations. What we see with the cloudlet is the ability to minimize query times in any scenario by monitoring independent parameters and intelligently delegating the projection subroutine to the proper device. Additionally, due the GPU implementation, the detection process in the cloudlet is significantly more robust than the mobile device, with the capacity to generate detection data for many faces per frame in a matter of milliseconds.

Meta-system performance is thus bottlenecked by the per-frame face density factor and the computational load of per-face projection. Therefore, the post-detection hardware-software implementation should utilize a parallel approach which will lend to the most observable evaluation criterion for the system: overall frame-rate. Our results show that, a cloudlet, equipped with suitable GPU accelerators can provide up to 128x speed improvement for the overall frame rate, although the degree of acceleration diminishes as the latency of the far cloud servers start dominating the run times.

Acknowledgment

This work was supported in part by the National Science Foundation grant CNS-1239423 and a gift from Nvidia Corp.

References

[1] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies, "The case for vm-based cloudlets in mobile computing," *Pervasive Computing, IEEE*, vol. 8, no. 4, pp. 14 –23, oct.-dec. 2009.

[2] J. Keller, "The atlantic: Cloud-powered facial recognition is terrifying," 2011, http://www.theatlantic.com/technology/archive/2011/09/cloud-powered-facial-recognition-is-terrifying/245867/.

[3] Microsoft Corporation, "Mobile Assistance Using Infrastructure (MAUI)," 2011, http://research.microsoft.com/en-us/projects/maui/.

[4] T. Soyata, *Enabling Real-Time Mobile Cloud Computing through Emerging Technologies*. IGI Global, Aug 2015.

[5] Y. Song, H. Wang, and T. Soyata, "Theoretical Foundation and GPU Implementation of Face Recognition," in *Enabling Real-Time Mobile Cloud Computing through Emerging Technologies*, T. Soyata, Ed. IGI Global, 2015, pp. 322–341.

[6] PlanetLab — An open platform for developing, deploying, and accessing planetart-scale services. http://www.planet-lab.org/.

[7] M. Kwon, Z. Dou, W. Heinzelman, T. Soyata, H. Ba, and J. Shi, "Use of network latency profiling and redundancy for cloud server selection," in *Proceedings of the 7th IEEE International Conference on Cloud Computing*, Jun 2014, pp. 826–832.

[8] E. Cuervo, A. Balasubramanian, D.-k. Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl, "Maui: making smartphones last longer with code offload," in *Proceedings of the 8th international conference on Mobile systems, applications, and services*, ser. MobiSys '10, New York, NY, USA, 2010, pp. 49–62. [Online]. Available: http://doi.acm.org/10.1145/1814433.1814441

[9] B.-G. Chun, S. Ihm, P. Maniatis, M. Naik, and A. Patti, "Clonecloud: elastic execution between mobile device and cloud," in *Proceedings of the sixth conference on Computer systems*, ser. EuroSys '11, New York, NY, USA, 2011, pp. 301–314. [Online]. Available: http://doi.acm.org/10.1145/1966445.1966473

[10] E. Chen, S. Ogata, and K. Horikawa, "Offloading android applications to the cloud without customizing android," in *Pervasive Computing and Communications Workshops (PERCOM Workshops), 2012 IEEE International Conference on*, march 2012, pp. 788 –793.

[11] T. Verbelen, P. Simoens, F. De Turck, and B. Dhoedt, "Cloudlets: bringing the cloud to the mobile user," in *Proceedings of the third ACM workshop on Mobile cloud computing and services*, ser. MCS '12, New York, NY, USA, 2012, pp. 29–36. [Online]. Available: http://doi.acm.org/10.1145/2307849.2307858

[12] T. Soyata, R. Muraleedharan, C. Funai, M. Kwon, and W. Heinzelman, "Cloud-Vision: Real-Time face recognition using a Mobile-Cloudlet-Cloud acceleration architecture," in *Proceedings of the 17th IEEE Symposium on Computers and Communications*, Jul 2012, pp. 59–66.

[13] C. Shi, M. H. Ammar, E. W. Zegura, and M. Naik, "Computing in cirrus clouds: the challenge of intermittent connectivity," in *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*, ser. MCC '12, New York, NY, USA, 2012, pp. 23–28. [Online]. Available: http://doi.acm.org/10.1145/2342509.2342515

[14] D. T. Hoang, D. Niyato, and P. Wang, "Optimal admission control policy for mobile cloud computing hotspot with cloudlet," in *Wireless Communications and Networking Conference (WCNC), 2012 IEEE*, 2012, pp. 3145–3149.

[15] T. Soyata, H. Ba, W. Heinzelman, M. Kwon, and J. Shi, "Accelerating mobile cloud computing: A survey," in *Communication Infrastructures for Cloud Computing*, H. T. Mouftah and B. Kantarci, Eds. IGI Global, Sep 2013, pp. 175–197.

[16] Y. Song, H. Wang, and T. Soyata, "Hardware and Software Aspects of VM-Based Mobile-Cloud Offloading," in *Enabling Real-Time Mobile Cloud Computing through Emerging Technologies*, T. Soyata, Ed. IGI Global, 2015, pp. 247–271.

[17] H. Wang, W. Liu, and T. Soyata, "Accessing big data in the cloud using mobile devices," in *Handbook of Research on Cloud Infrastructures for Big Data Analytics*, P. R. Chelliah and G. Deka, Eds. Hershey, PA, USA: IGI Global, Mar 2014, ch. 18, pp. 444–470.

[18] Amazon Web Services. http://aws.amazon.com.

[19] Microsoft Windows Azure. http://www.microsoft.com/windowazure.

[20] Google Cloud Platform. https://cloud.google.com/.

[21] E. Marinelli, "Hyrax: Cloud computing on mobile devices using mapreduce," Master's thesis, Carnegie Mellon University, Pittsburgh, PA, 2009.

[22] H. Ba, W. Heinzelman, C.-A. Janssen, and J. Shi, "Mobile computing - a green computing resource," in *Wireless Communications and Networking Conference (WCNC), 2013 IEEE*, Shanghai, P.R. China, 2013, pp. 4459–4464.

[23] T. Soyata, R. Muraleedharan, S. Ames, J. H. Langdon, C. Funai, M. Kwon, and W. Heinzelman, "COMBAT: mobile Cloud-based cOmpute/coMmunications infrastructure for BATtlefield applications," in *Proceedings of SPIE*, vol. 8403, May 2012, pp. 84 030K–84 030K.

[24] M. Hassanalieragh, A. Page, T. Soyata, G. Sharma, M. Aktas, G. Mateos, B. Kantarci, and S. Andreescu, "Health Monitoring and Management Using Internet-of-Things (IoT) Sensing with Cloud-based Processing: Opportunities and Challenges," in *2015 IEEE International Conference on Services Computing*, June 2015, pp. 285–292.

[25] O. Kocabas and T. Soyata, "Utilizing homomorphic encryption to implement secure and private medical cloud computing," in *IEEE 8th International Conference on Cloud Computing*, June 2015, pp. 540–547.

[26] N. Powers, A. Alling, R. Gyampoh-Vidogah, and T. Soyata, "AXaaS: Case for Acceleration as a Service," in *Globecom Workshops (GC Wkshps)*, Dec 2014, pp. 117–121.

[27] N. Powers and T. Soyata, "AXaaS (Acceleration as a Service): Can the Telecom Service Provider Rent a Cloudlet ?" in *4th IEEE International Conference on Cloud Networking*, Oct 2015.

[28] M.-H. Yang, D. Kriegman, and N. Ahuja, "Detecting faces in images : A survey," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 24, no. 1, pp. 34–58, 2002.

[29] P. Viola and M. J. Jones, "Robust real time face detection," in *Second International Workshop on Statistical and Computational Theories of Vision - Modeling, Learning, Computing, and Sampling*, July 2001, pp. 1–25.

[30] P. Viola and M. Jones, "Rapid object detection using a boosted cascade of simple features," in *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*, vol. 1. IEEE, 2001, pp. I–511–I–518 vol.1.

[31] P. Viola and M. J. Jones, "Robust real time face detection," *International Journal of Computer Vision*, vol. 57, no. 2, pp. 137–154, 2004.

[32] TMS320DM36x Digital Media System-on-Chip (DMSoC) Face Detection User's Guide. http://www.ti.com/lit/ug/sprugg8a/sprugg8a.pdf.

[33] Android API Reference. http://developer.android.com/reference/.

[34] OpenCV Library. http://opencv.org/.

[35] Intel Threading Building Blocks (Intel TBB). http://threadingbuildingblocks.org/.

[36] M. Turk and A. Pentland, "Eigenfaces for recognition," *Journal of cognitive neuroscience*, vol. 3, no. 1, pp. 71–86, 1991.

[37] S. Milborrow, J. Morkel, and F. Nicolls, "The MUCT Landmarked Face Database," *Pattern Recognition Association of South Africa*, 2010.

[38] Spacek, Libor, "University of Essex Face Data," http://cswww.essex.ac.uk/mv/allfaces/.