FaceDate: A Mobile Cloud Computing App for People Matching

Invited Paper

Pradyumna Neog
New Jersey Institute of Technology
University Heights
Newark, NJ 07102
pkn7@njit.edu

Hillol Debnath
New Jersey Institute of Technology
University Heights
Newark, NJ 07102
hd43@njit.edu

Jianchen Shan
New Jersey Institute of Technology
University Heights
Newark, NJ 07102
js622@njit.edu

Nafize R. Paiker
New Jersey Institute of Technology
University Heights
Newark, NJ 07102
nrp48@njit.edu

Narain Gehani
New Jersey Institute of Technology
University Heights
Newark, NJ 07102
gehani@njit.edu

Reza Curtmola
New Jersey Institute of Technology
University Heights
Newark, NJ 07102
reza.curtmola@njit.edu

Xiaoning Ding
New Jersey Institute of Technology
University Heights
Newark, NJ 07102
xiaoning.ding@njit.edu

Cristian Borcea
New Jersey Institute of Technology
University Heights
Newark, NJ 07102
borcea@njit.edu

Abstract

This paper presents FaceDate, a novel mobile app that matches persons based on their facial looks. Each FaceDate user uploads their profile face photo and trains the app with photos of faces they like. Upon user request, FaceDate detects other users located in the proximity of the requester and performs face matching in real-time. If a mutual match is found, the two users are notified and given the option to start communicating. FaceDate is implemented over our Moitree middleware for mobile distributed computing assisted by the cloud. The app is designed to scale with the number of users, as face recognition is done in parallel at different users. FaceDate can be configured for (i) higher performance, in which case the face recognition is done in the cloud or (ii) higher privacy, in which case the face recognition is done on the mobiles. The experimental results with Android-based phones demonstrate that FaceDate achieves promising performance.

CCS Concepts

•Human-centered computing →Mobile computing; •Computer systems organization →Cloud computing; •Software and its engineering →Middleware;

Keywords

Mobile cloud app, face matching

1 Introduction

Alice and Bob are sitting in nearby restaurants, and they do not know each other. Alice's favorite actor is Tom Cruise and would love to meet someone who looks like him. Bob, on the other hand, is keen on meeting someone like Shakira, and Alice resembles her. Bob also has facial similarity with Tom Cruise, and Alice might be interested in him. However, it is likely that they will never get a chance to meet. The outcome could be different if there was an app that could learn their face preferences and match them in real-time based on these preferences.

To the best of our knowledge, no current application can automatically solve this situation and suggest a location-based, *mutual match* in real-time to the two users. Many matching apps developed in the last decade [10, 16] use text-based profiles, which do not consider the face preferences of the users. Tinder [7], a newer and very successful app, provides options to filter out the results based on age-group, gender, and geographical proximity. In addition, it shows the photos of the matched users. However, it asks the user to decide if s/he likes the suggestions or not. This may be time consuming and, also, it does not ensure that the match is mutual. The work in [2] could be seen as an extension of Tinder, as the app trains on the user's photo choices over time. However, there is a high probability that the users will stop using the app if it does not provide any good recommendation in the initial stages. Also, showing user photos in situations where there is no mutual match could be considered a privacy problem.

This paper presents FaceDate, a mobile, location-based app that matches people based on their face preferences in real-time. The first novelty of FaceDate is that the users can train/re-train the app with photos of faces they like, and then the app can immediately provide good matches based on a face recognition algorithm. Unlike existing apps, FaceDate does not need potentially long time to learn the user preferences from the app usage. The second novelty of

BODYNETS 2016, December 15-16, Turin, Italy

DOI 10.4108/eai.15-12-2016.2267654

our app is that it can independently make a decision if two users satisfy each other's facial preferences and instantly inform them. FaceDate does not require manual input from the users during the match process. The third novelty is that, from a privacy point of view, FaceDate shares photos between users only if they mutually match. Therefore, user faces are not shown to people they do not like; thus, users can avoid potentially unpleasant interactions.

While our current design performs just face matching, FaceDate could be extended to use both face matching and traditional text-based profile matching.

We implemented a prototype of FaceDate on top of our Moitree [12] middleware for mobile distributed apps assited by the cloud. Moitree simplifies app development and provides cloud support for communication and computation offloading. In FaceDate, the users may choose to perform face recognition in the cloud for faster response time. If they have privacy concerns, they may choose to execute this operation on the mobile devices. In addition, Moitree provides support for dynamic group management, which allows FaceDate to adapt to users moving in and out of each other's proximity. For face recognition, FaceDate uses the Local Binary Patterns Histogram (LBPH) algorithm [11].

Experimental results over Android-based phones and Android-based virtual machines in the cloud demonstrate that FaceDate can provide good matches in a few seconds even when several users are searching for matches simultaneously in the same area. FaceDate scales with the number of users because the first phase of the face matching process, which involves potentially many users, is done in parallel at these users. This phase results in only a few good matches, and therefore, the sequential processing of the matches in the second phase (i.e., ensuring a mutual match) is bound by this low number of matches.

The rest of the paper is organized as follows. Section 2 provides an overview of FaceDate, followed by a brief description of Moitree, its supporting middleware, in Section 3. Section 4 presents FaceDate's design and implementation. Section 5 shows experimental results. Section 6 discusses related work, and the paper concludes in Section 7.

2 FaceDate Overview

The screenshots in Figure 1 show the steps taken by a FaceDate user to find a match. Figure 1a shows the "Profile Setup" screen. The user uploads a profile photo and provide basic information such as date of birth, gender, and a brief write-up about herself. Figure 1b shows the "Face Preference Setup" screen, where the user provides a set of preference photos. FaceDate will train itself automatically to perform matches on these photos. The "Search for Matches" screen, shown in Figure 1c, allows the user to start a search. FaceDate has a timeout (currently set to 5 seconds) to find a match; if a match request does not yield any result within this period, a *"No Match Found"* notification is displayed. If FaceDate is successful, the results are shown in the "Results" screen (Figure 1d). The user can then simply touch the photo, and the "Chat with Match" screen (Figure 1e) will appear.

Figure 2 provides an overview of the operation of FaceDate: the Querier starts the "Search" for a match; the two other users, Neighbor 1 and Neighbor 2, are in the neighborhood and participate in the search. The training of the app for each user happens before the search. When the search is activated, the Querier's device sends her profile photo to all FaceDate users in the neighborhood. The receiving devices execute a face recognition algorithm to try and match the received profile photo with their set of preference photos. In the figure, the Querier's profile photo matches only the preference photos of Neighbor 1. Thus, Neighbor 1 will respond with its profile photo, and the Querier will run the same face recognition algorithm on the response photo with respect to its preference photos. The profile photo of Neighbor 1 matches the preference photos of the Querier. Since it is a mutual match, FaceDate declares success and displays the profile photo of Neighbor 1 on the Querier's screen. The Querier then initiates a chat session with Neighbor 1.

While FaceDate has been designed to perform matching based only on facial features, it is straightforward to extend the app to work in conjunction with traditional text-based profile matching. The users could for example set a configuration parameter to choose between the two alternative matching processes.

Our implementation of FaceDate uses the Moitree [12] middleware that facilitates the execution of collaborative apps within groups of mobile users. Moitree has the following key features:

(1) High-level API for development of mobile collaborative apps assisted by the cloud.
(2) Seamless offloading and communication with app components in the cloud.
(3) Efficient user group management for location based apps, where the groups are dynamic over time and space.

These features are ideal for FaceDate since it is a collaborative location-based app and requires groups to be dynamically formed based on user location. The next section gives an overview of Moitree.

3 Moitree Overview

Avatar [8] is a mobile distributed platform which pairs each mobile device with a virtual machine (avatar) in the cloud. This setup supplements the limited resources available on the mobile with virtually unlimited resources in the cloud. However, it complicates the programming/execution of distributed apps, as they have to work over sets of mobile/avatar pairs.

Moitree is a middleware for the Avatar platform that facilitates the execution of collaborative apps within groups of mobile users. Each group is represented by a collection of mobile-avatar pairs. Groups can be context-aware, and Moitree takes care of their dynamic membership. For example, the group creation criteria can be based on location, time, and social ties.

Figure 3 shows a simple architectural view of Moitree. The middleware hides the low level details of the platform from programmers, and it provides a unified view of the mobile-avatar pairs by taking care of data synchronization, input abstraction, and computation and communication offloading. Moitree has APIs for managing groups and for abstract communication channels. All Moitree APIs are event-driven and callback-based. Moitree offloads user-to-user data communication to the cloud in order to save wireless bandwidth and improve latency. Its communication API provides four types of channels: ScatterGather, AnyCast, BroadCast, Point2Point to exchange data among group members.

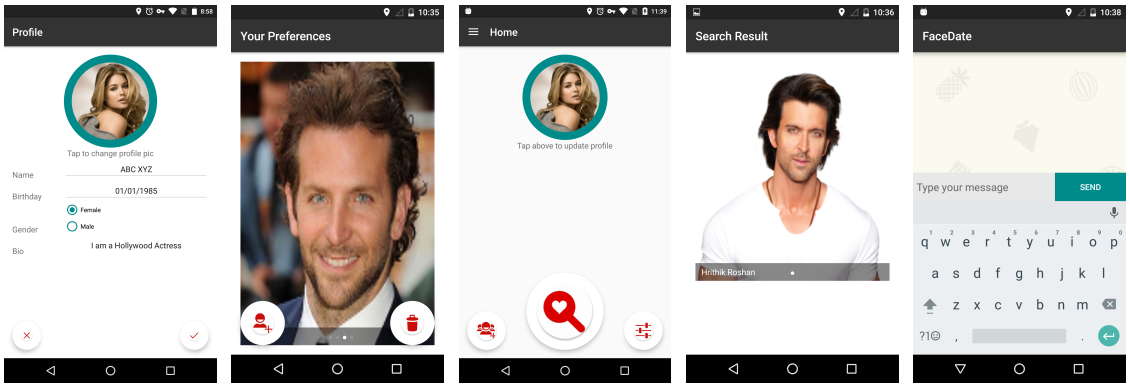


Figure 1: FaceDate Screen-shots: a)Profile Setup, b)Face Preference Setup, c)Search for Matches, d)Results, e) Chat with Match

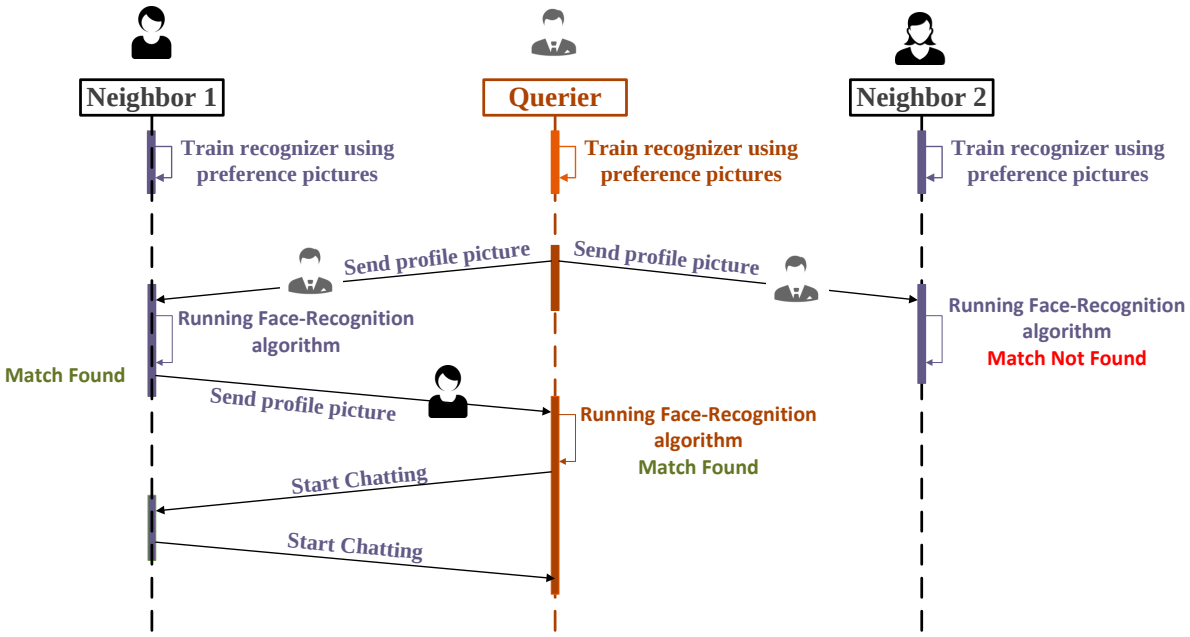


Figure 2: Overview of FaceDate Operation

4 Design and Implementation

Figure 4 shows the architecture of FaceDate. The *Search Manager* distributes the search requests from the user looking for a match to other users (i.e., to their FaceDate instances); it is also responsible for handling search requests from other users and delivering responses. The *Trainer* trains the app with preference photos of the user to facilitate face matching. The training results and the photos are stored in the *Photo Database*. The *Match Manager* matches photos by testing the similarity between the profile photos included in requests and the preference photos in the *Photo Database*. The *Search Manager* calls Moitree API functions to establish user groups for face matching and to enable communication among FaceDate instances at different user devices. Similarly, the *Chat Manager* uses the Moitree API for user-to-user communciation.

The pseudo-code in Algorithm 1 shows the workflow of matching in FaceDate, which consists of three major steps: (i) Location Updates, (ii) Profile and Preferences Setup, and (iii) Searching.

4.1 Location Updates

FaceDate needs to create location-based groups. Thus, it periodically updates the current location of the user by invoking the Google Location Services API [3] and notifies Moitree about location changes. The location is updated by a background thread initiated when the app is started (line 1 of Algorithm 1).

4.2 Profile and Preferences Setup

Before any search request can be handled, the user needs to setup her profile and add photos of her face preferences (lines 2 and 3). The app trains itself on the preference photos using the LBPH

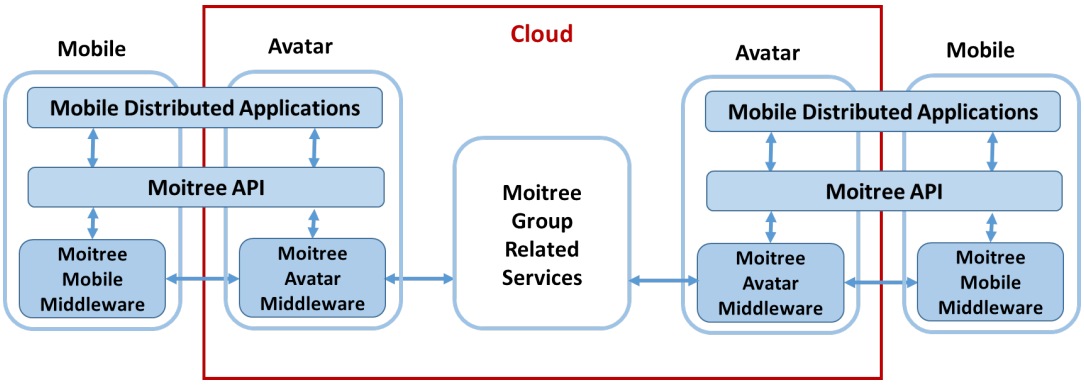


Figure 3: Architecture of Moitree Middleware

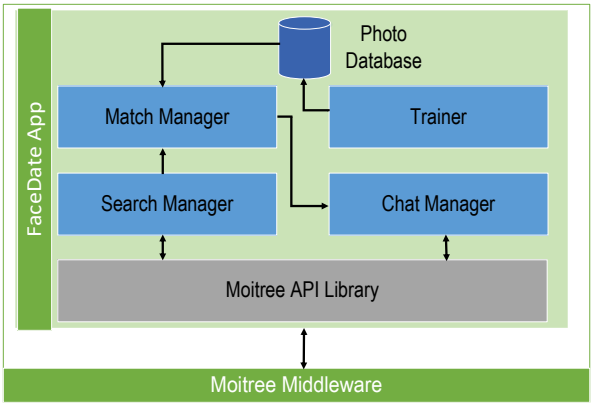


Figure 4: Architecture of FaceDate

algorithm provided by the JavaCV library [4], which is a wrapper to the OpenCV library [9].

FaceDate seeks similar looking faces rather than exact matches, within a user-specified threshold. To increase search accuracy, we have used three different settings of the LBPH algorithm: (1,8,8,8,60), (1,9,8,8,60) and (1,8,8,8,70). Each setting is a combination of radius, neighbors, grid x, grid y, and the threshold respectively. [1] We have also used a voting mechanism to reduce false positive results. For the same photo, if the results of at least two settings match, it is considered to match the preference photos.

4.3 Searching

In this step, FaceDate first calls the createGroup() method of Moitree to identify a *group* of users who are close to the geographical position of the querier. The search will be performed within the group. The invocation returns a "group" object in its callback method, as shown in line 4. Each member of the group (the querier and the participants) receives a group object as illustrated in lines 7 and 10. The callback methods dealing with the communication among the group members are setup earlier, as shown in line 5. The most important callback function handles query requests and responses,

[1]LBPH is a widely used algorithm for face matching, which is beyond the scope of this paper. Readers can access external sources for the algorithm and its settings.

Algorithm 1 FaceDate Matching Pseudo-Code

```
1: runThreadForLocationUpdates()      ▷ Thread to send
   location updates to Moitree
2: setupProfile()
3: setupPreferencePictures()
4: setupGroupCallback()
5: setupCommunicationCallback()
6: if querier == true then
7:    Group group = createGroup(RADIUS)
8:    group.distribute(querierProfilePicture)
9: else if participant == true then
10:   Group group = joinGroup()  ▷ Moitree pushes the
   group object
11:   getPersistentData()
12: end if
13: group.callBack(data){
14: if participant == true then
15:   querierProfilePicture = data.getPhoto()
16:   runFaceMatch(querierProfilePicture)
17:   if match == true then
18:      group.reply(participantProfilePicture)
19:   else
20:      discard()
21:   end if
22: else if querier == true then
23:   participantProfilePicture = data.getPhoto()
24:   runFaceMatch(participantProfilePicture)
25:   if match == true then
26:      displayResult()
27:      startChatManager() ▷ Prepares "Chat Manager"
   to handle chat
28:   else
29:      discard()
30:   end if
31: end if
32: }
```

and it is defined in lines 13-32. For a participant, the receipt of the group object also triggers the getPersistentData() function of Moitree (line 11) to retrieve any available persistent data from the group; if any data is received as a response, the app can process these data to check for a match.

Line 8 of Algorithm 1 describes how the querier uses the group object to forward the profile photo of the user to the other members of the group through the Scatter-Gather channel of Moitree. On receiving the profile photo, the participants run the face recognition algorithm to find a match as shown in lines 15-16. This process is

the same for the participants in the group, no matter whether they joined the group from the beginning or at a later stage. Let us note that Moitree seamlessly manages dynamic group joins and leaves.

If a match is found, the participant responds with its own profile photo using the same channel. Otherwise, it discards the message. This logic is shown in lines 17-21. The querier processes the response by running the face recognition algorithm to find a match with its own preference images. If a match is found on the querier's side as well, the profile photo in the response is then displayed to the querier as a "Match". Lines 22-31 cover the matching at the querier's side. The querier can select the photo displayed on the screen to start chatting with the other user (line 27). At this point, the *"Chat Manager"* is initialized to enable the chat through a Point2Point channel in Moitree.

5 Experimental Evaluation

This section evaluates the performance of the FaceDate prototype. The goals of the evaluation were to measure the time to train a dataset of preference photos, the time to match a photo against a trained dataset, and the end-to-end latency to perform matches.

We performed the experiments using multiple smart phones with different configurations: two Nexus 5, two Moto X, one Moto G3, and one Nexus 6. Each phone is paired with an avatar in an OpenStack-based cloud. The avatars are instantiated as Android 6.0 x86 VMs, configured with 4 virtual CPUs and 3GB memory. The Moitree group-related services are running on a Linux-based server in the cloud. The smart phones communicate with the avatars over the Internet, using WiFi.

FaceDate has two main phases:

- Training: This phase includes reading the preference photos, detecting and cropping faces, and training the LBPH recognizer for face recognition. The training phase is executed by all devices that run the FaceDate app.
- Matching: This phase includes sending the querier's profile photo, detecting and cropping the face from the profile photo received at participating devices, and performing face recognition using the LBPH recognizer.

We evaluated the performance of FaceDate when training and matching is done: (i) on the mobile devices, and (ii) at the avatar VMs. The first case corresponds to a setting in which users are concerned with privacy and store their profile photo and preference photos on their mobiles. The second case corresponds to a setting in which low latency is more important for users than privacy.

5.1 Performance of Training Phase

Figure 5(a) presents the total training time for each smart phone and the avatar as a function of an increasing training dataset. We varied the number of preference photos in the training dataset between 10 and 100. We used photos from several public datasets, and each photo was of a different individual. As expected, the training time on the avatar is substantially lower that the training times on the phones. We also observe significant differences between the smart phones, with the newer ones performing better. For all the phones and the avatar, the training time does not increase significantly up until 50 preference photos, but we notice a significant increase between 50 and 75 photos. We believe this is the result of Android's heap memory management mechanism used in the ART runtime: the increase in dataset size between 50 and 75 photos cause the ART VM to suddenly enlarge the heap to allocate more memory.

5.2 Performance of Matching Phase

Figure 5(b) presents the matching time for each smart phone and the avatar as a function of an increasing training dataset. As we increase the number of preference photos, the increase in matching time is substantially less steep than the increase in training time. When matching is performed on the avatar, the matching time remains almost constant. From the results, we can also see that there are some fluctuations in the matching time with the increase in the number of preference photos. We speculate that it is caused by the hardware and/or software optimization of the mobile devices rather than the matching operation.

The conclusion of these two experiments is that both phases should be performed at the avatars to achieve lower latency, if the users are not concerned about privacy. This alternative also saves energy on phones.

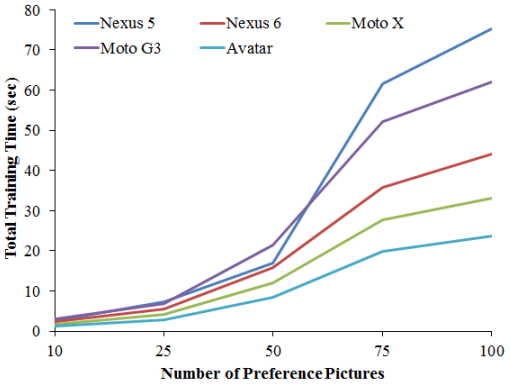
5.3 End-to-End Latency

In the following experiments, we measured the end-to-end latency for the FaceDate app. First, we used one querier and an increasing number of potential matching participants. The end-to-end latency is measured from the moment the querier starts the search until the querier finishes processing the received results. We configured this experiment such that, if there are n potential matching participants, all of them will return a match. The matching was performed on the phones to test the worst case scenario.

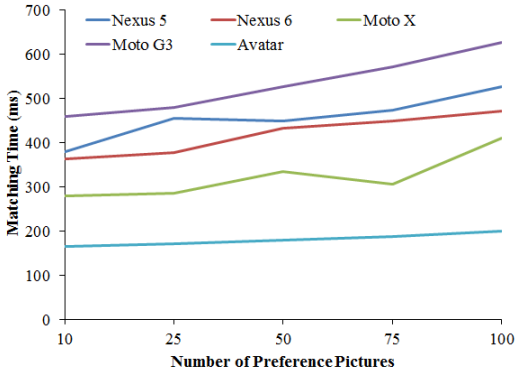
We used a Moto X phone as the querier and varied the number of participant phones. Figure 5(c) shows that the end-to-end latency increases almost linearly with the number of participants. On the other hand, the effect of increasing the number of preference photos is small. Several factors contribute to the increase in the end-to-end delay: (a) the slow response time on lower-end devices (such as the Moto G3 and Moto Nexus 5), and (b) the increasing number of matching operations on the querier device.

The next experiment investigated a more practical scenario where only one of the participants (among 5) will have a positive match result. This is to simulate a 20% positive match scenario, which is more reflective of FaceDate's performance in terms of real-life situations. Figure 5(d) shows that FaceDate is able to perform close to an ideal situation, where end-to-end latency does not vary almost at all. These results reflect the parallelism achieved by the app.

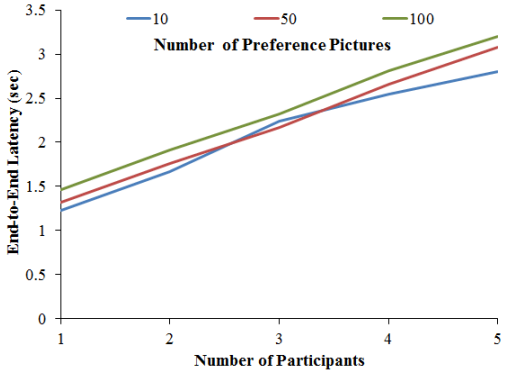
Finally, we evaluated the app to stress the system, where there is only one participant who responds positively and a varied number of queriers. Figure 5(e) shows the average latency for the queriers. In this experiment, each participant is a member of all the groups initiated by the queriers. The results for the "No Group Reuse" case show a significant increase in latency for each addition of a querier because a new group is created for each new query. To optimize on situations like this, when multiple users initiate the creation of different groups described by the same properties (e.g., location, time), Moitree can verify if a group with the same properties already exists and simply return a reference to this group. Thus, the overhead associated with group creation and management is



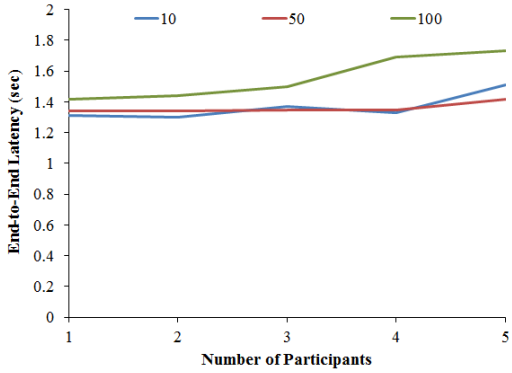
(a) Training time



(b) Matching time



(c) End-to-end latency for one querier when all responses are positive



(d) End-to-end latency for one querier when only one response is positive

(e) End-to-end latency with an increasing number of queriers and one positive responder

Figure 5: FaceDate Performance

drastically reduced. Therefore, the latency for the "Group Reuse" case is substantially lower.

6 Related Work

Social Matching [17] has been studied for a long time. The authors in [13] explored a theoretical framework of social, personal, and relational context for social matching by using interview studies. The work in [14] studied the essence of being familiar among strangers. SocialNet [16] tried to find people with common interests using social networks and co-location information collected by RF-based mobile devices. Serendipidy [10] used Bluetooth to find people in the proximity and then employed a profile database to suggest face-to-face interactions. Although these works identified common location as one potential matching criteria, they are mainly reliant on complex text-based profile matching. Similarly, popular online matching platforms [1, 5, 6] are also dependent on user-provided profiles and answers to questionnaires. In comparison, FaceDate uses an automated face matching algorithm to find matches, under

the assumption that appearance is generally the essential characteristic that makes people connect at the beginning of a relationship. Nevertheless, FaceDate could work in conjunction with profile-based matching.

Tinder [7] is another dating application, where the user can see the faces of all the nearby persons that match the filters set by the user. However, the results are not filtered on any facial characteristics. The user needs to manually select or discard images by browsing through all of them. In addition to being potentially tedious, this process does not guarantee that two people are mutually interested in each other. Furthermore, disclosing people's photos when they are not interested in each other could be viewed as a privacy-sensitive issue. FaceDate, on the other hand, provides only mutual matches to users; in this way, it simplifies user interaction and improves user privacy.

Similar to FaceDate, the work in [15] uses cloud and cloudlets to complete face recognition tasks faster. However, this solution works in an one-to-one scenario (mobile to cloud), without any collaborative computation. Photos taken by the mobile devices are pre-processed and sent to a centralized photo database for matching. In comparison, FaceDate works in a distributed manner, and the face recognition tasks can run on mobiles or be offloaded to the cloud depending on the user's privacy preferences.

7 Conclusion

This paper presented FaceDate, a mobile distributed app for location-based, real-time matching of people based on looks. FaceDate uses the Moitree middleware, which provides cloud support for the creation and maintenance of dynamic groups as well as abstract communication channels among mobile devices. The experimental results obtained using our Android-based prototype demonstrate end-to-end latency in the order of seconds and reasonable scalability. Therefore, FaceDate is practical in real-life scenarios. The app can be further improved through a hybrid search based on photos and profile text.

Acknowledgments

This research was supported by the National Science Foundation (NSF) under Grants No. CNS 1409523, SHF 1617749, CNS 1054754, DGE 1565478, and DUE 1241976, the National Security Agency (NSA) under Grant H98230-15-1-0274, and by the Defense Advanced Research Projects Agency (DARPA) and the Air Force Research Laboratory (AFRL) under Contract No. A8650-15-C-7521. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of NSF, NSA, DARPA, and AFRL. The United States Government is authorized to reproduce and distribute reprints notwithstanding any copyright notice herein.

References

[1] *eHarmony.* http://www.eharmony.com/.
[2] *Facedate.* https://angel.co/facedate.
[3] *Google Location Services.* https://developers.google.com/android/reference/com/google/android/gms/location/LocationServices.
[4] *JavaCV.* https://github.com/bytedeco/javacv.
[5] *Match.com.* http://www.match.com/.
[6] *okCupid.* https://www.okcupid.com/.
[7] *Tinder.* https://en.wikipedia.org/wiki/Tinder_(app).
[8] C. Borcea, X. Ding, N. Gehani, R. Curtmola, M. A. Khan, and H. Debnath. 2015. Avatar: Mobile Distributed Computing in the Cloud. In *2015 3rd IEEE International Conference on Mobile Cloud Computing, Services, and Engineering*. 151–156. DOI: http://dx.doi.org/10.1109/MobileCloud.2015.22
[9] I. Culjak, D. Abram, T. Pribanic, H. Dzapo, and M. Cifrek. 2012. A brief introduction to OpenCV. In *2012 Proceedings of the 35th International Convention MIPRO*. 1725–1730.
[10] N. Eagle and A. Pentland. 2005. Social serendipity: mobilizing social software. *IEEE Pervasive Computing* 4, 2 (Jan 2005), 28–34. DOI: http://dx.doi.org/10.1109/MPRV.2005.37
[11] K. Kadir, M. K. Kamaruddin, H. Nasir, S. I. Safie, and Z. A. K. Bakti. 2014. A comparative study between LBP and Haar-like features for Face Detection using OpenCV. In *2014 4th International Conference on Engineering Technology and Technopreneuship (ICE2T)*. 335–339. DOI: http://dx.doi.org/10.1109/ICE2T.2014.7006273
[12] M. A. Khan, H. Debnath, N. R. Paiker, N. Gehani, X. Ding, R. Curtmola, and C. Borcea. 2016. Moitree: A Middleware for Cloud-Assisted Mobile Distributed Apps. In *2016 4th IEEE International Conference on Mobile Cloud Computing, Services, and Engineering (MobileCloud)*. 21–30. DOI: http://dx.doi.org/10.1109/MobileCloud.2016.21
[13] Julia M. Mayer, Starr Roxanne Hiltz, and Quentin Jones. 2015. Making Social Matching Context-Aware: Design Concepts and Open Challenges. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems (CHI '15)*. ACM, New York, NY, USA, 545–554. DOI: http://dx.doi.org/10.1145/2702123.2702343
[14] Eric Paulos and Elizabeth Goodman. 2004. The Familiar Stranger: Anxiety, Comfort, and Play in Public Places. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '04)*. ACM, New York, NY, USA, 223–230. DOI: http://dx.doi.org/10.1145/985692.985721
[15] T. Soyata, R. Muraleedharan, C. Funai, M. Kwon, and W. Heinzelman. 2012. Cloud-Vision: Real-time face recognition using a mobile-cloudlet-cloud acceleration architecture. In *2012 IEEE Symposium on Computers and Communications (ISCC)*. 000059–000066. DOI: http://dx.doi.org/10.1109/ISCC.2012.6249269
[16] Michael Terry, Elizabeth D. Mynatt, Kathy Ryall, and Darren Leigh. 2002. Social Net: Using Patterns of Physical Proximity over Time to Infer Shared Interests. In *CHI '02 Extended Abstracts on Human Factors in Computing Systems (CHI EA '02)*. ACM, New York, NY, USA, 816–817. DOI: http://dx.doi.org/10.1145/506443.506612
[17] Loren Terveen and David W. McDonald. 2005. Social Matching: A Framework and Research Agenda. *ACM Trans. Comput.-Hum. Interact.* 12, 3 (Sept. 2005), 401–434. DOI: http://dx.doi.org/10.1145/1096737.1096740