FACE RECOGNITION USING DISTRIBUTED, MOBILE COMPUTING

Gregorio Hinojos and Phillip L. De Leon

New Mexico State University
Klipsch School of Electrical and Computer Engineering
Las Cruces, New Mexico, U.S.A.
ghhinojos@gmail.com, pdeleon@nmsu.edu

ABSTRACT

This paper describes a distributed computing framework called *BlueHoc*, that uses mobile devices connected using a Bluetooth, wireless ad hoc network. For a network composed of different devices, we have developed a load balancing method to optimize performance of BlueHoc. The eigenfaces technique for face recognition is implemented and used to benchmark performance. With four devices and an 80 subject face database, we can achieve a speedup (including all overheads) of $1.37\times$ without load balancing and with a 40 subject database a speedup of $1.08\times$ with load balancing. Because of fixed communications and initialization costs, the speedup factor can grow as the number of subjects in the recognition system grows. Consequently, by aggregating the computing capabilities of local mobile devices, BlueHoc provides an effective solution for distributed mobile computing.

Index Terms— Mobile computing, distributed computing, face recognition

1. INTRODUCTION

The pervasiveness and computational capabilities of mobile devices have grown at a remarkable rate in recent years [1–3]. The ubiquity of mobile devices presents an untapped resource that can be utilized to solve a variety of distributed computing problems which are impracticable to solve using a single mobile device or where access to conventional computing resources is limited. Current efforts towards combined mobile computing have focused on cloud computing systems, [4–6]. These systems are server dependent, where the server farms out work to the connected devices via WiFi.

Distributed computing architectures using the Bluetooth communication standard and mobile devices have been proposed [1–3]. In [1], the *BlueCube* protocol was proposed for constructing a hypercube structure using Bluetooth-connected mobile devices. BlueCube allows for automatic connectivity when at close proximity and is based on three phases: 1) ring construction which is an initialization phase, 2) scatternet construction which manages piconet connections and performs role switching to reduce the number of packet collisions, and 3) BlueCube construction which establishes a network structurally similar to a hypercube. The reults presented from BlueCube focused on packet transmission and collision and shortest and disjoint paths as opposed to computational performance.

In [3], the *BlueHydra* device communication paradigm based on Bluetooth was proposed and described. BlueHydra used the Marge Java Bluetooth framework for Bluetooth communication and JavaScript Object Notation (JSON) as a method invocation request and message response. Additionally, BlueHydra used the RFCOMM protocol for all Bluetooth connections. Performance results from BlueHydra are based on summing the elements of a floating-point array. Each device in the network receives a subarray whose size is equal to the size of the array divided by the number of network devices. BlueHydra's results with a five device network, showed a maximum speedup of $1.92\times$, when running 50,000,000 iterations of their computational function [3].

In [7], a distributed computing cluster composed of mobile devices running Debian Linux was constructed via USB and WiFi connections with a PC managing the cluster through a USB hub or a WiFi router. The LINPACK standard library was used to benchmark the cluster and the distribution was done via the Message Passing Interface (MPI) standard. The reported performance results reflected a close-to-linear scalability on floating point operations. However, no real-world application was implemented for benchmarking purposes.

In prior work, Hinojos, et. al. developed a distributed computing framework called *BlueHoc*, that uses Android mobile devices connected using a Bluetooth, wireless ad hoc network [8]. Unlike other distributed mobile computing systems in [4–6], BlueHoc is server-independent and creates a network with devices in close proximity. Similar to BlueCube, BlueHoc allows for role switching between master and slave nodes [1], however unlike BlueCube, BlueHoc has been implemented and evaluated on actual mobile devices as opposed to simulators. In contrast with BlueHydra, BlueHoc allows any device to act as master or slave in the computing network and ensures the system is functional in the event that a device drops from the network. BlueHoc is significantly different than the system presented in [7] in the following ways. First, BlueHoc runs on the popular Android OS which is commonly found on mobile devices. Second, BlueHoc communicates strictly over Bluetooth and is thus limited to devices in local proximity but does not require a centralized router or hub. Third, we benchmark our system using a real-world application, namely distributed face recognition.

In order to evaluate performance of BlueHoc, we have implemented an eigenfaces-based face recognition system which utilizes the distributed mobile computing environment to recognize image faces acquired on the master device in the network. Such an application and computing architecture could be useful in remote areas, such as border crossings or hostile areas where computing and network resources are limited, in order to identify subjects using only available, local, mobile computing resources.

This paper is organized as follows. In Section 2, we describe the BlueHoc distributed computing framework including information on system design, connection mechanisms, and message passing. In Section 3, we briefly describe the eigenfaces face recognition technique and in Section 4, we describe the initialization of the sys-

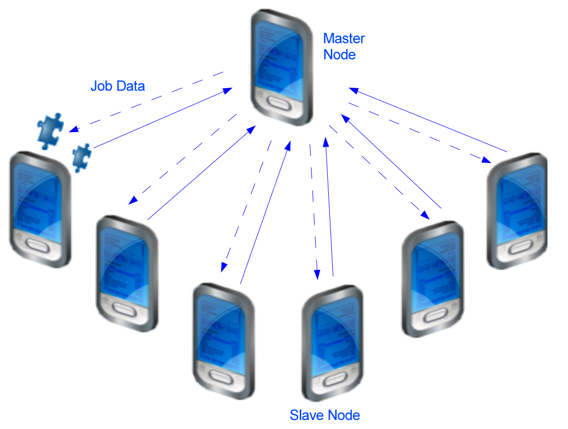

Fig. 1. In the BlueHoc architecture, the master transmits data subsets to all slave devices, makes a computation request, and combines slave results into a final result.

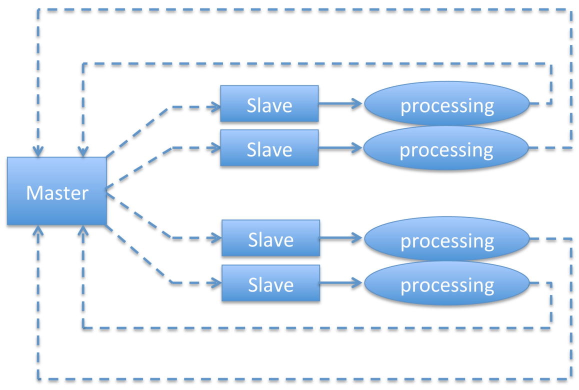

Fig. 2. In the BlueHoc message passing cycle, the master transmits data subsets to slave devices and makes a computation request. Once slave devices have completed the request each asynchronously transmits the intermediate result to the master which then computes the final result.

tem, distributed face recognition application, and load balancing. In Section 5, we describe the experimental setup and provide the benchmark results. Finally, in Section 6, we conclude the paper.

2. BLUEHOC

2.1. System Design

BlueHoc is based on a master/slave architecture where a master submits requests for computation to slave devices in the network as shown in Figure 1. A master device is one with more than one connection to other devices while slave devices are only connected to the master device. In the current implementation of BlueHoc, a single master device can connect with up to seven slave devices due to Bluetooth limitations [9]. The architecture is static in the sense that the master waits for all the slaves to join the network and the slave devices must remain connected throughout the work interval. Over time, slave devices may join or drop out of the network and BlueHoc will adjust each device's workload according to the size of the network. If the master device drops out, the network breaks apart and a new network could be constructed using an existing or new device as the master device.

A distributed computing job begins with the master partitioning and transmitting data subsets to all slave devices and making a computation request. Next, the data is processed by slave devices and results transmitted back to the master. Finally, the master combines intermediate results and produces a final result.

2.2. Server and Client Side Connection Mechanisms

When creating a connection, both server-side and client-side mechanisms must be implemented. On one side, the client or slave device initiates the connection using the server's or master's MAC address. On the other side, the server opens a socket and waits for a connection. The connection is authenticated from both sides using a universally unique identifier (UUID). The server begins by opening a `BluetoothServerSocket` and listening for incoming requests. A connection will only be accepted if a slave device requests a connection with a UUID matching the server UUID. In order for a client to initiate a connection, a remote `BluetoothDevice` is found by querying the list of paired devices or by scanning for available devices [10].

UUIDs are hard-coded in the BlueHoc framework so that they can be referenced by both server and client sides. In our implementation, seven UUIDs are created and stored in each device in order to establish the supported connections. Before a connection is made, the client selects a UUID when requesting a connection. When the server receives the request, it iterates through the possible UUIDs to find a match and establishes a connection.

2.3. Message Passing

Message passing allows processes to send/receive messages to/from other processes and includes method invocation, system information, and data packets. In BlueHoc, messages are passed asynchronously from master to slaves and vice-versa, but messages are not passed between slaves. Computation across BlueHoc is conducted synchronously meaning that the master sends a method invocation to the slaves requesting computation and the master waits until all slaves have completed the computation request and returned the intermediate result. Figure 2 illustrates the message passing cycle.

In BlueHoc, messages sent from the master to the slaves contain function calls to be executed. Messages are "active" in the sense that they are capable of doing processing on their own since each message has an identifier to be used by the handler to determine the function to be executed upon message arrival [11]. Furthermore, computation of the active message is performed using the data in the message as well as data stored in the receiving node (slave). These messages are restricted to the single program, multiple data (SPMD) programming model, where the sender and receiver know what data can be accessed at the receiving node [11].

3. FACE RECOGNITION APPLICATION

To demonstrate the utility of BlueHoc in a real-world application, we have developed a face recognition application. In face recognition, features are extracted from a person's reference image(s) and stored. In order to recognize an unknown person from a face image, features are extracted and compared to the reference and the closest match determines the identity of the unknown person in the test image [12]. One of the most popular and relatively simple face recognition techniques is the *eigenfaces* method [12–15].

The eigenfaces method consists of learning and classifying stages. In the learning stage, for each person's reference images, we construct an image vector by stacking all columns of the image matrix, subtract the mean vector (average face image) from each image vector, and compute the covariance matrix of the vectors. Next, eigen-analysis of the covariance matrix is performed and the eigenvectors associated with the D largest eigenvalues, called *principal components*, are retained [12]. Finally, the mean-centered image vectors are projected onto the eigenspace spanned by the principal components.

In the classifying stage, an image is acquired of an unknown person, the columns of the image matrix are stacked, the average face image is subtracted, and the resulting vector is projected onto the eigenspace. The euclidian distance is then calculated from the projected test image to all training images in the database. The image with the closest distance is selected and the person is identified [12].

4. FACE RECOGNITION OVER BLUEHOC

4.1. Distributed Face Recognition

Before any work is done, the BlueHoc system must be initialized. System initialization consists of executing the learning stage on any device containing the reference images. A `facedata.xml` file is created which contains the projection matrix, mean vector, and reference image vectors and this file is propagated to all other devices.

Distributed face recognition using BlueHoc begins with the master device acquiring a face image. The master then preprocesses the image, which consists of creating a bounding box around the face, cropping the image, converting to black and white, and performing histogram equalization. The processed image is projected to the face space and then transmitted to the slave devices along with a job request. Included in the job request are the indices to the subset of reference image vectors that the slave scores against via a distance calculation. Each slave device returns the best match (based on minimum distance) from their respective subsets and the master selects the best overall match from the minimum distance of the intermediate (slave) results.

4.2. Load Balancing

In our initial implementation of face recognition over distributed, mobile devices, each data subset contained the same number of reference images. However, for a heterogeneous network of slave devices with unequal computing capabilities, the load is unbalanced and the overall compute time is constraint by the slowest device. In our load-balanced implementation, the device's computing capabilities are known by the master ahead of time and are factored into the range of reference image vector indices each slave scores against. In this way, the computational load is balanced accordingly and the overall computation time is minimized compared to the unbalanced implementation.

A hash table was created containing the devices' MAC addresses and processing rates. Before each job request, the master looks up the processing rate for the ith slave device in the network using the MAC address, and determines the number of reference images n_i that are to be processed,

$$n_i = \frac{\Delta T}{T_i} \cdot \frac{M}{N}, \quad 1 \leq i \leq N \tag{1}$$

Mobile Device	Processor (quad-core)	t_s
Asus Transformer	1.2 GHz Tegra 3 T30L	4.117
Nexus 7	1.3 GHz Tegra 3	3.593
Samsung Galaxy Note2	1.6 GHz Cortex-A9	2.959
Samsung Galaxy S3	1.4 GHz Cortex-A9	3.503

Table 1. Android devices used in BlueHoc. t_s is the average time for a single device to recognize a subject based on a 40 subject face database.

where T_i is the ith device's processing rate, ΔT is average processing rate for all devices, M is the number of face images in the database, and N is the number of devices. The device's processing rate is predetermined by clocking the average time per distance calculation.

5. EXPERIMENTAL RESULTS

5.1. Experimental Setup and Baseline Results

The devices used on BlueHoc consist of two smartphones and two tablets (listed in Table 1) all running the Android OS (v4.0 or v4.1). The designated master was the Samsung Galaxy S3. To ensure consistent benchmarking results, all wireless services except Bluetooth were disabled, all non-essential processes were closed[1], and all UI process indicators were disabled. The native Java function, `System.currentTimeMillis()` was used for timing based on the master device's clock.

We use the AT&T Cambridge faces database a.k.a. the ORL Database of Faces [16] which consists of 10 different face images for each of 40 different subjects. By duplicating the set of images and renaming the subjects, we created a larger database of 80 subjects. For each trial, we randomly chose seven images per person for training and three images per person for testing. Classification accuracy for the 40 subject database, averaged over four devices and ten trials, was 96% which is consistent with results reported in literature [12–14, 17].

In order to measure the baseline performance of the face recognition system using a single mobile device, a test face image is displayed on a computer monitor, the device captures the image via the device's camera, performs the computation required for recognition, and decides the identity. We do not include image acquisition time in the overall measure.

Using one of the test images, we clocked the recognition time for each trial for various database sizes, M. The average time per recognition, t_s averaged over 10 trials serves as the reference or baseline time. For each device, the average time per recognition for a 40 subject database, t_s is shown in Column 3 of Table 1. The average recognition time for a single device (averaged over the various devices) as a function of database size is shown with the magenta line in Figure 3.

5.2. Benchmark Results using BlueHoc

In order to measure the performance of the distributed face recognition system using BlueHoc, the system follows the event sequence

[1]Some applications such as Samsung's backup, suggest, and voice apps are pre-installed and could not be closed. Additionally, pre-installed device-usage monitoring apps are also not able to be completely closed. Based on our observations, these applications have negligible impact on the results.

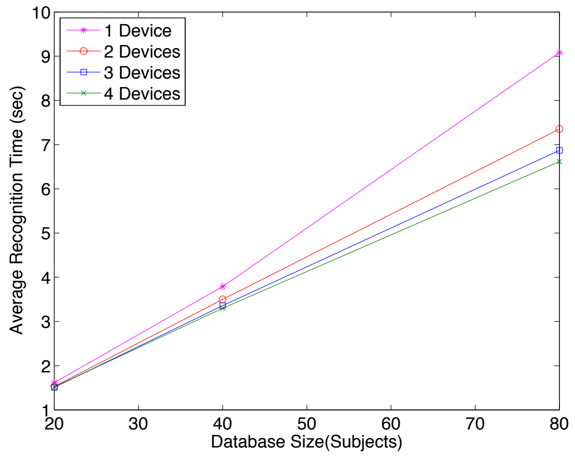

Fig. 3. Average recognition time for BlueHoc versus a face database with 20, 40, and 80 subjects. As the database size grows, additional devices lower the recognition time despite transmission overheads.

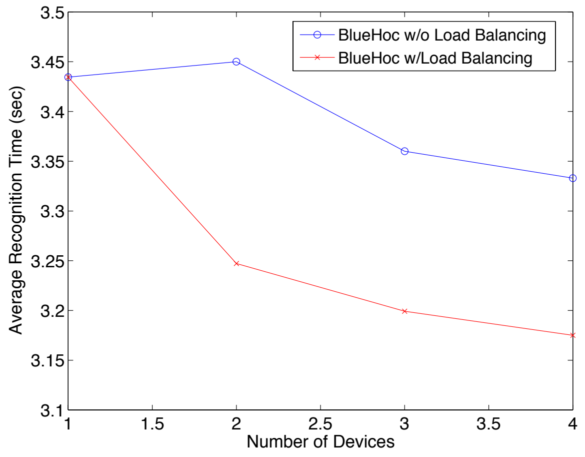

Fig. 4. Average recognition time of BlueHoc with and without load balancing as a function of the number of devices in the network. Load balancing improves BlueHoc performance.

described in Section 4.1. We clock the time from the acquisition of the face image by the master device, image preprocessing, projection to face space, transmission of feature vector to slave devices, feature vector scoring, transmission of identity with minimum distance by slave devices, and the overall identification by the master device based on minimum distance of identities. The average time per recognition over 10 trials for the distributed system is denoted t_d and the speedup gain with the distributed system compared to the average single device is

$$G \quad = \quad t_s/t_d. \tag{2}$$

Figure 3 shows the average recognition time for BlueHoc consisting of various numbers of devices versus a face database with 20, 40, and 80 subjects. The results were averaged using all possible combinations of devices (${}_NC_4$ where N is the number of devices in the network out of the four possible devices listed in Table 1). Compared with a single device, the two-, three-, and four-device distributed computing network shows a 1.24×, 1.32×, 1.37×, respectively speedup for the 80 subject database. The three- and four-device networks show only small gains due to communications overheads but clearly illustrate that as the face database size grows, additional devices can increase performance.

Figure 4 shows the average execution time for BlueHoc consisting of various numbers of devices versus a face database with 40 and 80 subjects subjects with load balancing. The results were averaged using all possible combinations of devices. Comparing Figure 3 with Figure 4, we see that load balancing further decreases average recognition times.

Regarding Figures 3-4, we further observe the following. First, as the face database size increases, the potential for speedup increases because of significantly increased required computation. Second, on small-size databases, as the network size increases the speedup does not improve and potentially decreases due to fixed communications overheads. Third, as the size of the database increases, it is beneficial to add devices to the network, conversely when database is small there are only small gains (if any) to be had from additional devices.

6. CONCLUSIONS

We have developed and implemented a framework for distributed mobile computing over Bluetooth, adhoc networks called BlueHoc. Such a system may be useful in remote areas where computing and communications resources are limited. We have tested BlueHoc using a face recognition application and benchmarked the system for different sizes of face databases and number of devices in the network. With four devices and an 80 subject face database, we can achieve a speedup (including all overheads) of 1.37× without load balancing and with a 40 subject database a speedup of 1.08× with load balancing. We have also demonstrated that as the face database size grows, the speedup factor can grow due to fixed communications and initialization costs.

7. REFERENCES

[1] C. Chang, C. Chang, and J. Sheu, "BlueCube: Constructing a hypercube parallel computing and communication environment over Bluetooth radio systems," *J. Parallel Distrib. Comput.*, vol. 66, no. 10, pp. 1243–1258, Oct. 2006.

[2] R. Shepherd, J. Story, and S. Mansoor, "Parallel computation in mobile systems using Bluetooth scatternets and Java," in *Proc. Int. Conf. Parallel and Distrib. Comput. and Networks*, 2004.

[3] M. Gartrell, J. Kelly, and S. Razgulin, "BlueHydra: Distributed Computing on Mobile Bluetooth-Enabled Devices," Tech. Rep., Department of Computer Science, University of Colorado, Bouler, CO., 2008.

[4] J. Shafer, S. Rixner, and A.L. Cox, "The hadoop distributed filesystem: Balancing portability and performance," in *Perf Anal. of Syst. Soft. (ISPASS), IEEE Int. Symp. on*, 2010, pp. 122–133.

[5] D. Huang, X. Zhang, M. Kang, and J. Luo, "Mobicloud: Building secure cloud framework for mobile computing and communication," in *Service Oriented System Engineering (SOSE), IEEE Int. Symp. on*, 2010, pp. 27–34.

[6] E. Marinell, “Hyrax: Could computing on mobile devices using MapReduce,” M.S. thesis, School of Computer Sience, Carnegie Mellon University, Pittsburgh, PA, 2009.

[7] F. Busching, S. Schildt, and L. Wolf, “Droidcluster: Towards smartphone cluster computing – the streets are paved with potential computer clusters,” in *Distrib. Comput. Syst. Workshops (ICDCSW), Int. Conf. on*, 2012, pp. 114–117.

[8] G. Hinojos, C. Tade, S. Park, D. Shires, and D. Bruno, “Bluehoc: Bluetooth ad-hoc network android distributed computing,” *Int. Conf. on Parallel and Distrib. Process. Tech. and Appl. (PDPTA)*, Jul. 2013.

[9] “`http://www.bluetooth.com/`,” Feb. 2013.

[10] “`http://developer.android.com/index.html`,” Feb. 2013.

[11] T. von Eicken, D. Culler, S. Goldstein, and K. Schauser, “Active messages: a mechanism for integrated communication and computation,” *ISCA Annual Int. Symp. on Computer Archit.*, vol. 19, pp. 256–266, 1992.

[12] M. Turk and A. Pentland, “Face recognition using eigenfaces,” *Proc. IEEE Computer Society Conf. Computer Vision and Pattern Recognition*, pp. 586–591, 1991.

[13] J. Zhang, Y. Yan, and M. Lades, “Face recognition: Eigenfaces, elastic matching, and neural nets,” *Proc. IEEE*, vol. 85, no. 9, Sep. 1997.

[14] A Tolba, A El-Baz, and A El-Harby, “Face recognition: A literature review,” *Int. J. Signal Process*, vol. 2, pp. 88–103, 2006.

[15] K. Kyungnam, “Face recognition using principle component analysis,” in *Int. Conf. on Computer Vision and Pattern Recognition*, 1996, pp. 586–591.

[16] F. Samaria and A. Harter, “Parameterisation of a stochastic model for human face identification,” in *Proc. IEEE Workshop on Appl. of Computer Vision*, Dec. 1994, pp. 138–142.

[17] R. Jafri and H. Arabnia, “A survey of face recognition techniques,” *J. Info. Process. Syst.*, vol. 5, no. 2, pp. 41–68, Jun. 2009.