Cloud-Vision: Real-time Face Recognition Using a Mobile-Cloudlet-Cloud Acceleration Architecture

Tolga Soyata*, Rajani Muraleedharan*, Colin Funai*, Minseok Kwon†, Wendi Heinzelman*

*Dept. of Electrical and Computer Engineering
University of Rochester
Rochester, NY 14627
{soyata,rmuralee,funai,wheinzel}@ece.rochester.edu

†Dept. of Computer Science
Rochester Institute of Technology
Rochester, NY 14623
jmk@cs.rit.edu

***Abstract*—Face recognition applications for airport security and surveillance can benefit from the collaborative coupling of mobile and cloud computing as they become widely available today. This paper discusses our work with the design and implementation of face recognition applications using our mobile-cloudlet-cloud architecture named MOCHA and its initial performance results. The challenge lies with how to perform task partitioning from mobile devices to cloud and distribute compute load among cloud servers (cloudlet) to minimize the response time given diverse communication latencies and server compute powers. Our preliminary simulation results show that optimal task partitioning algorithms significantly affect response time with heterogeneous latencies and compute powers. Motivated by these results, we design, implement, and validate the basic functionalities of MOCHA as a proof-of-concept, and develop algorithms that minimize the overall response time for face recognition. Our experimental results demonstrate that high-powered cloudlets are technically feasible and indeed help reduce overall processing time when face recognition applications run on mobile devices using the cloud as the backend servers.**

I. Introduction

In our daily lives, face recognition applications that automatically identify an individual from captured images or videos are everywhere, for applications such as surveillance, airport security, law enforcement, and border patrol. Face recognition algorithms analyze images, extract information such as the shape, size and position of the facial features (e.g., eyes, nose, mouth), and then use these extracted features to search a facial database to locate matching images. The algorithms of highest accuracy (e.g., over 90%) typically require intensive computation [1].

Another interesting trend is the plethora of lightweight mobile devices available today, such as tablets, netbooks and smartphones. These devices are becoming increasingly powerful, with more processing power, storage, and sensing capabilities. In addition, it is now possible to rent computing, storage, and network resources as needed via *cloud computing*, in which data is processed and stored remotely at large-scale compute and data centers [2], [3], [4]. The ability of mobile devices to access cloud computing resources is expected to support a myriad of new applications including augmented reality, high-performance file systems, image processing (e.g., 2D to 3D transformation), secure data management and our application of interest, real-time face recognition.

While there are many face recognition applications that will benefit from the collaborative coupling of mobile and cloud computing, one in particular is an extension of Amber Alerts to mobile phones. In this scenario, a central authority (e.g., the FBI) would extend their Amber alerts such that all available cell phones in the area where a missing child was last seen that opt-in to the alert would actively capture images and perform face recognition. Due to the significant amount of processing required to perform face recognition, as well as the need for a large database of images with which to compare the captured faces in images taken by the cell phones, this application is simply not possible using the mobile devices' compute power alone, requiring access to cloud computing.

This paper describes our work with the design and implementation of face recognition on the MOCHA (MObile Cloud Hybrid Architecture) cloud computing platform, which provides a mobile-cloudlet-cloud architecture [5]. One of the well-known challenges for using the cloud as a server is the long latency between the mobile device and the cloud server in comparison to localized computing and small-scale distributed computing called cloudlet [6] . Given this challenge, our primary focus is on evaluating the performance of face recognition algorithms using our MOCHA architecture with a focus on the overall response time as well as validating the system functionalities when request is sent from the mobile device . Our specific research question is how to distribute computing load in order to achieve the minimal response time given diverse communication latencies and server computing powers when mobile devices interact with multiple servers in the cloud. We use smartphones as our main mobile device to capture images and to forward them to the cloudlet; the cloudlet performs computation on the received images and finds matching images from the database in collaboration with the cloud.

Our contributions in this paper are summarized as follows:

1) We utilize a mobile-cloudlet-cloud framework and develop algorithms that minimize the overall response time for face recognition based on estimated communication latencies and processing powers of the cloud.
2) We demonstrate that high-powered cloudlets are technically feasible and provide benefit to mobile device face recognition applications, among others. To the best of our knowledge, no prior work has yet shown this in large-scale with specific architectures, algorithms, and

978-1-4673-2713-8/12/$31.00 ©2012 IEEE

applications, although some initial ideas were introduced in the literature.

The rest of the paper is organized as follows. In Section II, we describe our MOCHA architecture, providing the details of each component. In Section III, we show analytically the benefit of smart partitioning of the computation among the cloudlet and cloud servers, for a generic job composed of multiple independent tasks. In Section IV, we provide an overview of the face recognition algorithm and its structure in MOCHA. In Section V, we present results from our experiments, and in Section VI an overview of related work is given. Finally, we summarize our conclusions and future work in Section VII.

II. The MOCHA Architecture

Despite the one order of magnitude higher computational power of today's mobile devices compared to just a few years ago, the relative computational power ratio of a non-mobile and a mobile device will stay approximately the same for the foreseeable future, since the architectural and technological improvements are applied to both mobile platforms as well as desktop platforms simultaneously. The approximate relative computational parameters for the mobile devices and a large cloud operator, such as Amazon AWS, as well as both a home-based and an enterprise *cloudlet*, are shown in Table I. As described in the Microsoft MAUI project [7], some applications might never be feasible from mobile devices, due to the high latency mobile-cloud connection as well as the complexity of managing the multiple potential cloud servers. However, adding a *cloudlet*, a local device that provides 100 to 1000 times higher computational power with minimal latencies, creates possibilities for running latency sensitive and computationally-intensive applications such as face recognition from a mobile device.

	Mobile	Home-CL	Ent-CL	Cloud
Comp Power	1	100 - 10K	10K - 100K	10K - 100M
RAM	1	10 - 100	100 - 1K	1K - 10M
Storage	1	100	10K	100K - 1M
Comm Delay	1	10	10	1000

TABLE I
Comparison of normalized compute-capabilities and communications delays of a mobile device, cloudlet (home- and enterprise-grade), and the cloud.

As a solution for this mobile face recognition problem, we propose using the MOCHA architecture, illustrated in Figure 1. Using MOCHA, mobile devices such as smartphones, touchpads, and laptops are connected to the cloud (e.g., Amazon Web Services [2], Windows Azure [3]) via a cloudlet, a dedicated server designed from commodity hardware supporting multiple network connections such as 3G/4G, Bluetooth, WiFi and Internet. The cloudlet determines how to partition the computation among itself and multiple servers in the cloud to optimize the overall quality of service (QoS), based on estimates of the QoS metrics (e.g., latency, cost) over the different links/routes and of the servers. We will discuss each component of the MOCHA architecture in detail in this section.

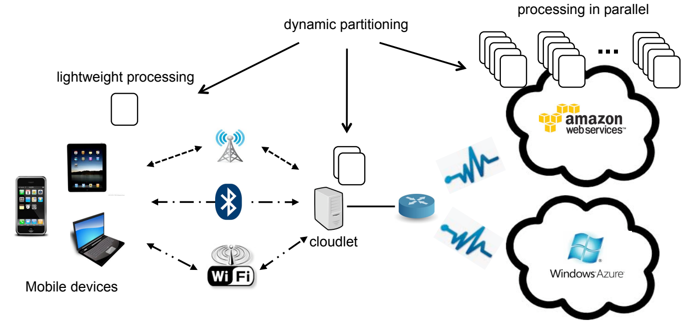

Fig. 1. The MOCHA Architecture: mobile devices interact with the cloudlet and the cloud via multiple connections and use dynamic partitioning to achieve their QoS goals (e.g., latency, cost).

A. Mobile Device

In our work, we assume mobile devices, such as smart phones and iPads. The main task of our mobile device is to acquire images and send them to the cloudlet in raw form for pre-processing. If the cloud was near the mobile and the communication latencies were negligible, the captured image(s) could be sent to the cloud and a program could be run to perform real-time template matching over a large database located in the cloud. However, sending the raw data (e.g., image(s)) to the cloud is very network-intensive and may be unnecessary, since the cloud might only require a very small subset of the information (e.g., Haar features and classifiers). Thus pre-processing of the image(s), either at the mobile device or at the cloudlet, is necessary. After face recognition is complete, the mobile device receives the results back from the cloudlet or directly from the cloud (or, in the case of the Amber Alert application, the results are sent to an appropriate FBI location). This entire process is performed transparently so that the mobile user does not see any difference compared with performing local computing at the mobile device.

B. Cloudlet

In our architecture, the cloudlet is a special-purpose inexpensive compute-box with the capability of massively parallel processing (MPP) [8], using GPUs such as an Nvidia GT520 [9]. Our concept cloudlet has 150 single Gflop of compute capability, 2 GB memory, and 40 W total power consumption with a cost under $100. The cloudlet, equipped with a GPU and a lightweight CPU (e.g., INTEL ATOM [10]) is still necessary to run the serial applications, such as the OS kernel, as well as the algorithms described in Section III which require high MPP power.

C. Cloud

Cloud computing provides computing and storage resources remotely in a pay-as-you-go manner. In our system, a client program running on the cloudlet (or the mobile device if directly accessing the cloud) sends a request to the servers on Amazon AWS where the actual program runs on virtual instances in parallel, and the results are sent back to the requester (mobile device or cloudlet). We have simulated the cloud using our internal heterogeneous compute cluster of 14 computers.

III. Algorithmic Optimization for MOCHA

Cloud computing is based on the fundamental concept of sharing resources among locally and globally available cloud servers to improve QoS and application performance. Due to the availability of dynamic computing public and private infrastructures, an optimal approach to partitioning computation/tasks to servers that balances performance goals such as response time and reliability is required. Many applications that benefit from using the cloud have real-time constraints, with the speed of response being the driving factor for choosing either global or local resources. Performance factors such as processing time and communication latency directly influence the speed of a cloud server's response to requests for computation on data. For example, our measurement data shows that average propagation delays to the AWS data centers in Virginia, Oregon and Singapore are 110, 226 and 595 msec, respectively. These delays inflate to approximately 2, 6 and 18 seconds when the mobile device sends a 420 KB image file to the servers. Thus, it is important to understand how the response time is impacted by different scenarios, including increased number of available cloud servers, changing processing times of the cloud servers, and varying communication latencies, as well as the impact of using the cloudlet.

Assuming a processing job consisting of multiple independent tasks, we consider two approaches for partitioning the computation (tasks) among the available cloud servers and the cloudlet, assuming identical tasks. (1) Fixed: the tasks are equally distributed among the available cloud servers (or the cloudlet). The total response time is the time that it takes for the last response to be returned. (2) Greedy: we first order the servers (and the cloudlet) by their (known) response times, and give the first task to the server (cloudlet) that can complete this task in the minimum amount of time. We then give the second task to the server (cloudlet) that can complete this task in the minimum amount of time (note that this may be the same server as given the first task if the time for the first server to complete both tasks one and two is less than the time for the second server to complete just task two). We continue in this way, using a greedy approach to select the server (cloudlet) for each task in turn. The overall response time is again the time it takes for the last response to be returned. This is the lower bound of response time.

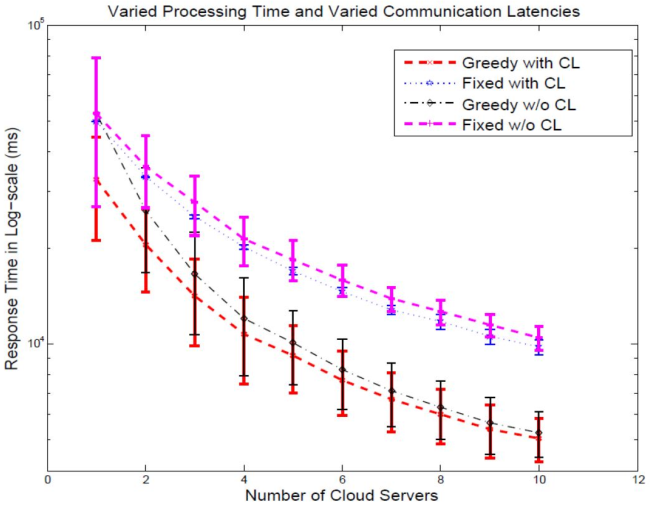

Fig. 2. Simulated response time using varied processing times and communication latencies for 10 cloud servers.

The response times of these two approaches are compared using Monte Carlo simulation, where a computing job consisting of 1000 identical tasks is distributed among a number of cloud servers with varied processing capabilities and communication latencies and the cloudlet. In the first set of simulations, the processing time of each cloud server is a fixed value, chosen from a uniform distribution between 10 and 100 ms to complete each task, while the processing time of the cloudlet is set to 100 ms to complete each task. The latency for the communication from the cloudlet to each cloud server is also a fixed value, chosen from a uniform distribution between 100 ms and 1 s to send a packet to the cloud server or from the cloud server back to the cloudlet. We run 100 simulations where we choose different random processing times and latencies for each of the cloud servers. In each run, we simulate between 1 and 10 cloud servers to assign the 1000 tasks according to the algorithm (Fixed or Greedy). We also include results using the Fixed and Greedy algorithms when the cloudlet is not available for processing the data (i.e., the mobile sends the data directly to the cloud servers). As expected, the response time of the Greedy approach is the lowest with or without the cloudlet, providing as much as 45% and 41% improvement in response time as shown in Figure 2. Thus a-priori knowledge of the cloud servers' (cloudlet) processing times and communication latencies enables a large speed-up in response time, given heterogeneous cloud servers and communication latencies. We can also see the benefit to using the cloudlet when smart partitioning (Greedy) is used, providing improvements up to 16% compared to the Greedy approach when the cloudlet is not used.

The fixed and greedy approaches have similar performance when all the cloud servers have the same processing times and communication latencies, and thus, in this scenario, a smart partitioning approach is unnecessary. However, network conditions vary dynamically, and factors like resource reliability and system downtime are unpredictable. For example, recently, due to an Amazon EC2 outage, cloud computing businesses like BigDoor were disrupted [11]. In such a dynamic environment as the Internet, all available cloud servers may not have the same communication latencies. If we can learn about the current conditions, we can use our optimal (greedy) approach for partitioning the tasks. To demonstrate further the advantage of smart partitioning when the latencies to different cloud servers vary, we simulated a scenario with 10 cloud servers whose processing times were all set to 1 ms per task, with the cloudlet processing time set to 1 ms per task. The communication latencies with the cloudlet varied from no difference (all

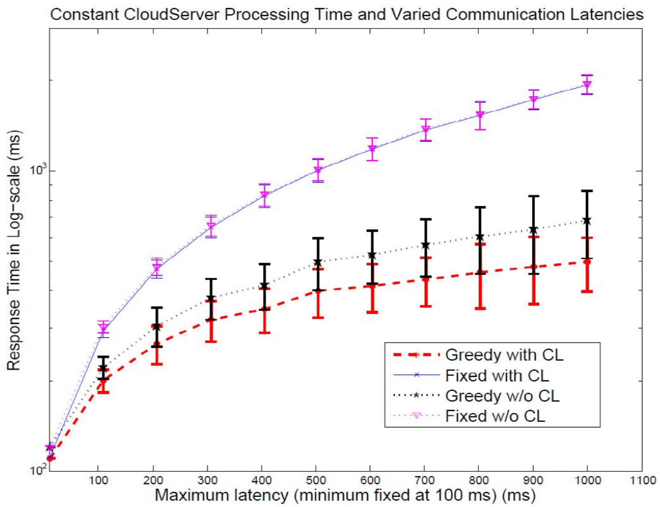

Fig. 3. Simulated response time using using a fixed processing time at each cloud server and varying communication latencies.

latencies set to 10 ms) to an increasingly more heterogeneous environment with latencies chosen from a uniform distribution between 10 ms and an upper limit that varied from up to 1 s. Once again, we consider the case with and without a cloudlet available. Figure 3 shows the response of the Greedy and Fixed algorithms in log-scale. As clearly shown in this figure, the benefit of smart partitioning is most pronounced when the variations in cloud server communication latencies are high. Therefore, knowledge of available communication resources can greatly speed up the response time for cloud computing applications. However, it would be difficult for a mobile device to determine these latencies, especially given their dynamic nature. A cloudlet when used as an intermediary (as in the MOCHA architecture) is crucial for enabling the estimation of link latencies and hence realizing these clear benefits of smart partitioning. Additionally, these results show that a speed-up of approximately 8% is possible when using a cloudlet compared with no cloudlet.

IV. Use of MOCHA Architecture for Cloud-based Face Recognition

The previous section provided motivation for us to partition computation among the cloudlet and our cloud servers. In this section, we look specifically at the computation required by our Cloud-Vision face recognition application and then explore how these computations can be partitioned among the cloud servers and the cloudlet. Cloud-Vision is executed in two separate phases: 1) face detection, which uses the widely accepted Viola-Jones algorithm [12] that progressively narrows an initial large set of face location candidates to the final set of detected face locations; and 2) face recognition, which uses the classic Eigenfaces approach, initially introduced in [13] to determine the likelihood of each detected face matching one of the template images in a database. These two phases of the overall Cloud-Vision concept require significant computation, as explained in detail in the following subsections, and hence require cloud resources to perform the face recognition application for a mobile device.

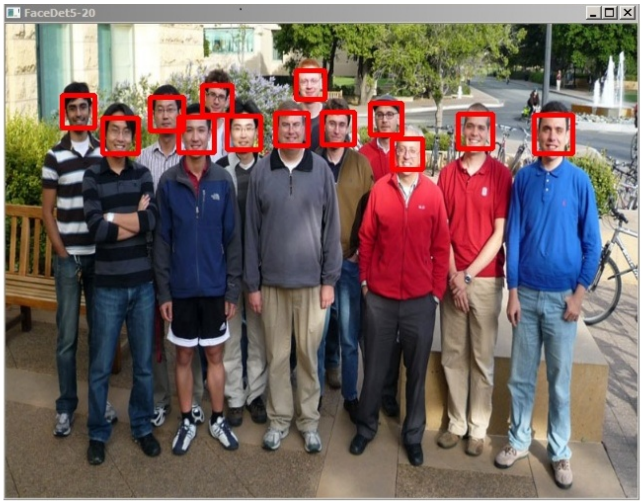

Fig. 4. Face Detection: This phase determines potential locations for faces. This version was run using a 20x20 initial box and a 5% progressive increment. Accuracy is around 99%.

A. Face Detection

As shown in Figure 4, the face detection phase of the overall Cloud-Vision process determines the potential locations of the human faces within an image. We have utilized the Haar Features and Haar Classifiers described in [14] to perform face detection. This iterative approach begins with fairly primitive classifiers that group potential face candidates based on a small number of features. These simple classifiers in this initial stage have low computational complexity but must operate on a large amount of data, and they produce a large number of face candidates. The algorithm then progressively eliminates some of these candidates by using increasingly more sophisticated classifiers based on additional features at successive stages of the detection pipeline, such that the final stage outputs the detected faces with high confidence. Although the number of remaining candidates is significantly less at each successive stage, the complexity of the calculations increases at almost the same rate, and thus the overall computational complexity of each pipeline stage of this detection algorithm stays somewhat constant. In our implementation, we use a 32-stage pipeline.

B. Face Recognition

The face recognition phase of the overall Cloud-Vision process determines the match-likelihood of each face to a template element from a database. The potential locations of the faces determined in the previous face detection phase are fed into this phase for recognition. The recognition algorithm yields one of a few potential results for each face candidate determined by the detector: (1) not a face, (2) a face, but not in the database, and (3) a face and in the database. We have employed the widely-accepted Eigenfaces approach [13], which calculates an orthogonal set of M Eigenfaces for a given training set of N faces, where $M \ll N$. Figure 5 shows the 29 Eigenfaces calculated for a set of 500 images (i.e., $N = 500$ and $M = 29$). Thus each face from the original N faces can be represented as a point within the M-dimensional space spanned by the M Eigenfaces. This permits

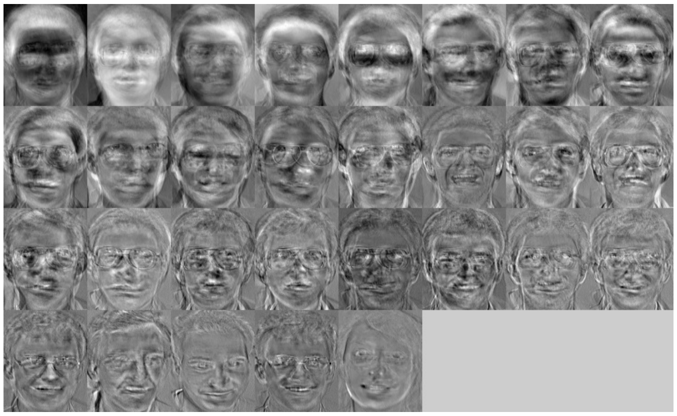

Fig. 5. Face Recognition: A set of M orthogonal face matrices (called Eigenfaces) are used to represent the original images, permitting significant reduction in computation recognition. 29 Eigenfaces shown in this figure are calculated for the original 500 face templates.

a significant reduction in the amount of computation that has to be performed to recognize a face within a given database, as well as a significant reduction in the amount of storage required for the template images (database).

To recognize a face, the algorithm simply transforms the face into a point within the M-dimensional space spanned by the Eigenfaces and calculates the Euclidean distances between the point of the face to be detected and the points of all template faces from the database. If the Euclidean distance is above a large threshold, the algorithm determines that the potential face is actually not a face (outcome (1)), meaning that the detection algorithm yielded a false positive. Otherwise, if the Euclidean distance is above a small threshold to all the template faces, the algorithm determines that the face is not in the database (outcome (2)). In this case, if desired, this face can be added to the database by re-calculating the Eigenfaces and the Eigenfaces representation of the newly introduced face and updating the databases of all the cloud servers. If the Euclidean distance to one of the template faces is below a small threshold, the algorithm detects a match for the face (outcome (3)).

C. Cloud-Vision: Partitioning Computation/Communication

Based on the previous sections, we observe that the Cloud-Vision approach has a large amount of computation, thus necessitating cloud computing resources. Additionally, this computation can be partitioned among mulitple cloud servers to speed up the response time. In our implementation we assume that the mobile device, which has very limited compute power, simply captures the image and sends this off for face recognition. While the mobile device could directly send the image to the cloud, this would require the mobile to coordinate the computation partitioning, as well as the communication with the various cloud servers. Given the limited compute power of the mobile device, coupled with the potentially large latency of the mobile-cloud server links, this process can be improved with the use of the MOCHA architecture, which provides a cloudlet for coordinating the face detection and face recognition phases. Additionally, given the relatively greater compute power of the cloudlet, the cloudlet can provide some initial pre-processing of the image such that less data need to be sent to the cloud servers for performing the face detection and face recognition tasks. Although it is possible to model the compute nodes and the corresponding latencies as a graph [15], [16], [17], permitting more sophisticated graph-based synchronization algorithms, this is left as future work.

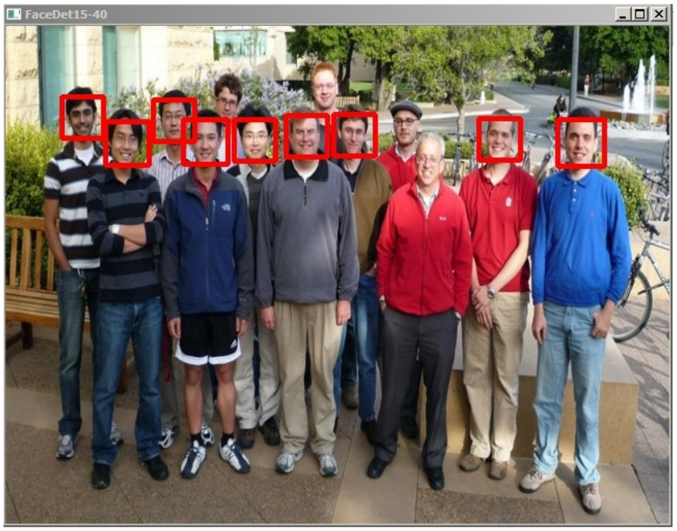

Fig. 6. Face Detection: Same program run with an initial box size of 40x40, and a progressive increment ratio of 15%. Although this version demands 10x lower compute-power, there is a dramatic decrease in accuracy.

We perform different parallelizable stages of the face detection/recognition algorithm at a different cloud server (or the cloudlet). We assume that each cloud server knows the classifier functions it must perform (i.e., the training phase of the algorithm to produce the appropriate classifier functions [12], [14], [18], [19] is an offline process). Given the linear nature of the classifier functions, once part of the data has been classified by one stage, the results of the classification can be sent to the next stage at a different cloud server, and then this cloud server can begin its stage of classification on the data already completed by the previous stage. This process continues until the cloud server performing the final stage of the face detection algorithm, which sends the results back to the cloudlet. In this way, large speed-ups in the detection process are possible even though the face detection stages are pipelined. Clearly, it is most beneficial if the stages are run on cloud servers close to each other such that the communication latency is low.

Upon receipt of the locations of the detected faces from the final cloud server, the cloudlet extracts the detected faces from the original image and sends each face to a different cloud server to perform face recognition. We assume that all cloud servers have the template database with the Eigenfaces and the representation of all template faces. Thus, only the face image to be recognized must be sent to each cloud server. For a very large template database, the cloud server can choose to only perform recognition among a portion of the face templates, and send the face image to other cloud servers to perform recognition among a different portion of the face templates. This may provide some speed-up in the face recognition process depending on communication latencies. All results of the face recognition algorithm are eventually sent back to the cloudlet, which are, then, aggregated and sent back to the mobile device.

Although fully-dynamic partitioning is possible, distributing the face detection stages and the face recognition tasks to cloud servers based on the runtime characteristics of the communication latencies, this is beyond the scope of this paper. We have assumed fixed stages and placed them in different cloud servers a-priori, such that, based on the structure of the cloud servers, the best overall response time is achieved. While the detailed evaluation of the utility of the cloudlet and the response times expected from different architectural configuration of the mobile, cloudlet and the cloud servers will be provided in the following section, an immediate positive outcome of adding the cloudlet layer can be observed from Figure 6. This figure shows the face detection results with an initial box size of 40x40 and a progressive increase of 15% at each step. Compared to Figure 4, which has an improved detection accuracy due to the initial box size of 20x20 and progressive increment ratio of 5%, Figure 6 depicts a drastic reduction in detection accuracy (i.e., only 9 out of 13, or 69%, of the faces are detected). Both results were obtained using a 500-face database with an overall response time of 300 ms. The only difference between the two figures is the addition of the cloudlet as the intermediate layer, coordinating the scheduling and performing part of the vital tasks. Despite the 10x compute-demand required to get the results in Figure 4, the cloudlet was able to buffer a significant portion of the high latencies among the mobile and the cloud servers, thereby achieving a significant improvement in detection accuracy. Although more sophisticated synchronous timing algorithms can be utilized for the overall timing by using retiming techniques [20], [21], this is left as future work.

V. Performance Evaluation

A. Experimental Setup

Our experimental hardware platform is a distributed heterogeneous cluster of 13 servers, workstations and a laptop, running either Windows 2008 Server or Windows 2007 Professional. These computers are distributed over three separate geographic locations, connected through a high-speed broadband link (5/35 Mbps upstream/downstream). We have traced the connections and found the links to contain approximately 8 to 10 hops with a round-trip ping latency of 20 to 40 ms.

Our development platform is Microsoft Visual Studio 2010 C++, and Open CV libraries [18] for the development of our face detection and face recognition programs. The program we have developed resides in every member of the cluster, assuming either the cloudlet or cloud-server responsibility. There is only one cloudlet in our setup and multiple cloud servers. To narrow the scope of this extensive development process, we have made some simplifications:

1) Since we are assuming that, the mobile is only responsible of sending the raw image (in compressed JPG format) to the cloudlet, this process is not repeated during our experiments. Constant mobile transfer time is added in every experiment. 2) Rather than specific hardware, we have only used one CPU thread to emulate the cloudlet. 3) We have restricted the number of geographical locations to three, which is sufficient to prove our concept. 4) We have left the usage of GPUs as future work, as it is not needed to prove our concept of MOCHA.

Our cloud / cloudlet software sends request and response packets directly from the cloudlet emulator to the cloud servers. The cloudlet code is by-passed to emulate the scenario when there is no cloudlet. This prohibits any speed-up to be gained from utilizing the cloudlet, such as running intelligent scheduling algorithms, or performing a portion of the work without dispatching to the cloud. To emulate the speed of a mobile device, we have run the same cloudlet code in a loop of 10x, which is consistent with the relative mobile/cloudlet compute-power ratios reported in Table I.

UNIVERSITY	**CPU**	**Speed**	**C/T**	**CompPow**
PC1	i7-990x	3.73GHz	6C/12T	1.00
PC2	DX48BT2	2.83GHz	4C/4T	0.49
PC3	i7-960	3.2GHz	4C/8T	0.68
PC4	i7-930	2.8GHz	4C/8T	0.60
PC5	i7-2620M	3.4GHz	4C/8T	0.61
PC6	i7-960	3.2GHz	4C/8T	0.68
PC7	i7-960	3.2GHz	4C/8T	0.68
OFFSITE 1	**CPU**	**Speed**	**C/T**	**CompPow**
PC8	i7-980x	3.6GHz	6C/12T	0.97
PC9	i7-2600	3.4GHz	4C/8T	0.67
PC10	i7-2600	3.4GHz	4C/8T	0.67
OFFSITE 2	**CPU**	**Speed**	**C/T**	**CompPow**
PC11	i7-980x	3.6GHz	6C/12T	0.97
PC12	i7-930	3.8GHz	4C/8T	0.60
PC13	i7-930	2.66GHz	4C/8T	0.60

TABLE II
Hardware used for the experiments. PC1 at the university also emulates the cloudlet. CPUs used in the servers are reported in terms of their cores/threads (C/T) and their normalized compute-powers (CompPow).

Table II shows the list of our equipment and their geographical distribution. The main location, our university, contains the laptop, cloudlet emulator, mobile emulator, and a majority of the cloud servers. The other locations were accessed using Microsoft Remote desktop to run the software we have developed. Our software uses request and response TCP/IP packets for communication, similar to MPI. Each packet contains a task-code for a task (e.g., calculate Haar classifier number 45). Each response packet contains the answer to its corresponding request packet. Our results have been reported by utilizing timers placed inside the code.

B. Experimental Results

Figure 7 shows the response times for the Fixed and Greedy algorithms using no cloudlet and an emulated cloudlet. As shown in this figure, performing sophisticated task partitioning and a portion of the task demands significantly higher compute-power from the mobile device then is available. This is due to the two to five order of magnitude compute-power difference between the mobile and the cloudlet as shown in Table I. When the number of cloud servers increases, the

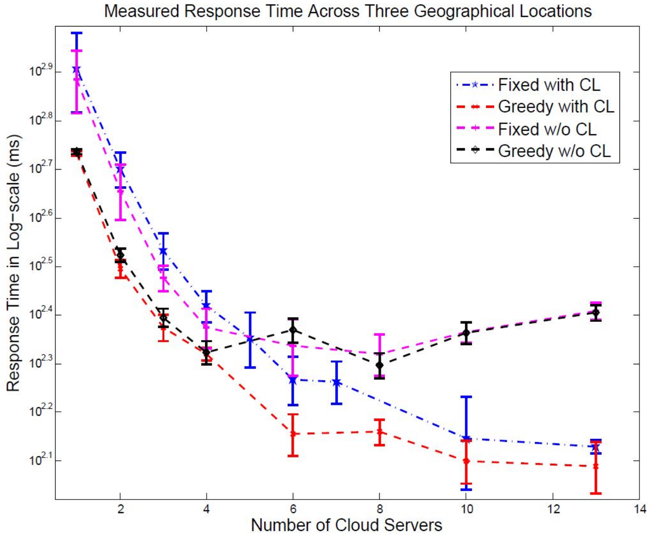

Fig. 7. Measurement of the response times using fixed and greedy approaches. Cloudlet and no-cloudlet scenarioes are also depicted.

performance in fact decreases due to the extreme compute-strain on the mobile device. A state-of-the-art mobile device contains a dual-core 1GHz ARM-based CPU as of today with a 1W power budget. Alternatively a cloudlet with a 50W power budget and cost of under $100 can be two to three orders of magnitude more efficient compared to the mobile device. This difference is further emphasized since this extreme compute-demand over-taxes the mobile device as the workload increases (e.g., excessive threading will cause cache memory thrashing and excessive context switching, causing a steep performance drop-off). To prove this concept, we observe this performance inflection point at above four cloud servers.

As shown in Figure 7, no such slowdown is evident when a cloudlet is used. Owing to its two order-of-magnitude higher power budget, the cloudlet can successfully schedule tasks to more than ten cloud servers on different CPU threads, send and receive packets from the Internet on different CPU threads, and perform a portion of the tasks by itself during idle cycles. By intelligently offloading the necessary tasks from the mobile device, the cloudlet serves as a compute and/or communications buffer for the mobile device. This synergistic coupling of the mobile-cloudlet dramatically improves the overall response time by permitting the mobile device to perform tasks that are only possible to be performed by a 50W desktop device.

Using the Greedy algorithm (which is more realistic since some degree of run-time information is typically available, permitting this approach), we observe a near-2x speed-up from Figure 7 when the cloudlet is utilized with 13 cloud servers. Alternatively, this difference is smaller when four or fewer cloud servers are used. This result is consistent with our simulations in Section III for low server counts. However, the difference becomes larger since our simulations did not take into account the compute-strain in the mobile device. In case of the high server count, it becomes a daunting task for the mobile device to capture images from the camera, dispatch them to multiple cloud servers, and process the results that are received, thereby causing the performance inflection point.

VI. Related Work

The rapid advancements in mobile technology and cloud computing have enabled many mobile cloud computing applications. Irrespective of the security concern presented in [22], where face recognition and cloud computing is used to match a snapshot with a person's online identity in less than a minute, the smartphones's resources and cloud computing techniques needs to be exploited to benefit applications that utilizes Big Data. In [23] Hyrax enables smartphone applications to form a distributed computing cloud, which implements Hadoop [24] in Android phones using MapReduce function. Similarly, Huerta-Canepa and Lee [25] developed ad-hoc cloud computing architecture motivated by smartphone's ubiquitousness and available resources. Chun and Maniatis [26] investigate the feasibility of dynamically partitioning application processing between weak devices (i.e., smartphones) and clouds. They formulated this partitioning problem as an optimization that minimizes execution time given resource constraints. Inspired by the fact that the data access patterns of many mobile applications depend on the current location of the user, Where-Store [27] caches cloud data on the phone based on location information using the current location and the prediction of future locations.

Energy efficiency of smart mobile devices is studied in conjunction with cloud computing in [28]. RACE [29] proposes the idea that mobile phones can act as data relay nodes to augment network connectivity between servers and the battery-constrained mobile devices. Motivated by the fact that virtual data centers in the cloud partition compute power but have little control over network bandwidth partitioning, Seawall [30] explores possibilities that clients share network resources in such a way that each of them isolate themselves from others fairly. The software architecture in smartphones in support of secure personal clouds is discussed in [31]. CasCap [32] is a cloud-assisted context-aware power management framework that utilizes the resources in the cloud to support secure, low-cost and efficient power management for mobile devices. Gilbert et al. [33] propose a visionary system that automatically validates how secure mobile apps are at app markets using cloud computing. In [34] Berriman et al used Montage image mosaic engine to compare the cost and performance of processing images on the Amazon EC2 cloud and Abe high performance cluster at National Center for Supercomputing Applications (NCSA) to emphasize the necessity of provenance management.

In this paper we analyze the different communication strategies chosen to achieve performance like response time and cost for face recognition application using comprehensive algorithm that can accelerate computation and communication within the cloud. In addition, we investigate whether high-powered cloudlets are technically feasible and cost-effective using GPUs for efficient mobile-cloud interactions.

VII. CONCLUSIONS AND FUTURE WORK

We have presented a mobile-cloudlet-cloud architecture called MOCHA as a platform for our target face recognition application. This architecture is designed to minimize the overall response time of the face detection and face recognition algorithms given heterogeneous communication latencies and compute powers of cloud servers at diverse geographical placements. We have designed MOCHA to integrate mobile devices (e.g., smartphones), the cloudlet, and multiple cloud servers and demonstrated that cloudlets are technically feasible and beneficial at minimal additional costs. We have used an intuitive barrier based synchronization for utilizing multiple cloud servers for parallelism. To our knowledge, this is the first work to show such an architecture with the three components working together with specific algorithms, applications, and initial results. Our simulation results show that 1) more intelligent task partitioning algorithms employed by the cloudlet permits response-time improvement by offloading work from the mobile device, 2) the response time decreases as the number of cloud servers increase and this improvement is more emphasized when cloudlets are in place, and 3) communication latencies affect the response time considerably, which can be partially coalesced when cloudlets are used as buffers. Our experimental results validate the simulation results and show that MOCHA indeed reduces the overall response time for face recognition. We plan to extend our experiments using real cloud services (e.g., AWS) and mobile devices (e.g., Android phones) with more heterogeneous latencies and compute powers in large-scale. Our future work also includes more sophisticated synchronization algorithms permitting cloud-to-cloud communications, rather than multiple cloudlet-cloud communications links.

REFERENCES

[1] Ming-Hsuan Yang, David Kriegman, and Narendra Ahuja, "Detecting faces in images : A survey," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 24, no. 1, pp. 34–58, 2002.

[2] Amazon, "Amazon Web Services (AWS)," http://aws.amazon.com.

[3] Microsoft, "Windows Azure," http://www.microsoft.com/windowazure.

[4] Google, "Google App Engine," http://code.google.com/appengine.

[5] T. Soyata, R. Muraleedharan, J. H. Langdon, C. Funai, M. Kwon, and W. B.Heinzelman, "Mobile cloud-based compute/communications infrastructure for battlefield applications," in *SPIE Defense, Security, and Sensing 2009. Modeling and Simulation for Defense Systems and Applications VII*. 2012, vol. 8403, SPIE.

[6] Mahadev Satyanarayanan, Paramvir Bahl, Ramón Cáceres, and Nigel Davies, "The case for VM-based cloudlets in mobile computing," *IEEE Pervasive Computing*, vol. 8, no. 4, pp. 14–23, Oct. 2009.

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

[8] David B. Kirk and W.-M. Hwu, *Programming Massively Parallel Processors*, Morgan-Kaufmann, 2010.

[9] Nvidia Corp, "GeForce GT 500 Series," 2011, http://en.wikipedia.org/wiki/GeForce_500_Series.

[10] Intel Corp, "Atom In-Order Processor," http://en.wikipedia.org/wiki/Intel_atom.

[11] Aislyn Greene, "Amazon: Some data lost in cloud outage is not recoverables," 2011.

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

[13] Matthew Turk and Alex Pentland, "Eigenfaces for recognition," *Journal of Cognitive Neuroscience*, vol. 3, no. 1, pp. 71–86, 1991.

[14] Paula Viola and Michael 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.

[15] Tolga Soyata, Eby G. Friedman, and J. H. Mulligan, "Integration of clock skew and register delays into a retiming algorithm," in *Proceedings of the International Symposium on Circuits and Systems*, May 1993, pp. 1483–1486.

[16] Tolga Soyata, Eby G. Friedman, and J. H. Mulligan, "Monotonicity constraints on path delays for efficient retiming with localized clock skew and variable register delay," in *Proceedings of the International Symposium on Circuits and Systems*, May 1995, pp. 1748–1751.

[17] Tolga Soyata and Eby G. Friedman, "Synchronous performance and reliability improvements in pipelined asics," in *Proceedings of the IEEE ASIC Conference*, Sep 1994, pp. 383–390.

[18] Gary Bradski and Adrian Kaehler, *OpenCV Computer Vision with OpenCV Library*, O'Reilly, 2008.

[19] Junguk Cho, Shahnam Mirzae, Jason Oberg, and Ryan Kastner, "Fpgas," in *Elseiver Science*, Feb 2009, pp. 1–18.

[20] Tolga Soyata and Eby G. Friedman, "Retiming with non-zero clock skew, variable register and interconnect delay," in *Proceedings of the IEEE Conference on Computer-Aided Design*, Nov 1994, pp. 234–241.

[21] Tolga Soyata and Eby G. Friedman, "Incorporating Interconnect, Register and Clock Distribution Delays into the Retiming Process," *IEEE Transactions on Computer Aided Design of Circuits and Systems*, vol. CAD-16, no. 1, pp. 105–120, Jan 1997.

[22] 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/.

[23] E. Marinelli, "Hyrax: Cloud Computing on Mobile Devices using MapReduce," Tech. Rep. CMU-CS-09-164, Carnegie Mellon University, September 2009.

[24] Hadoop Project Management Committee, "Hadoop," 2008–present, http://hadoop.apache.org.

[25] G. Huerta-Canepa and D. Lee, "Ad Hoc Virtual Cloud Computing Providers for Mobile Devices," in *Proceedings of ACM Mobile Cloud Computing and Services (MCS)*, June 2010.

[26] B. Chun and P. Maniatis, "Dynamically Partitioning Applications Between Weak Devices and Clouds," in *Proceedings of ACM Mobile Cloud Computing and Services (MCS)*, June 2010.

[27] P. Stuedi, I. Mohomed, and D. Terry, "WhereStore: Location-based Data Storage for Mobile Devices Interacting with the Cloud," in *Proceedings of ACM Mobile Cloud Computing and Services (MCS)*, June 2010.

[28] A. Miettinen and J. Nurminen, "Energy Efficiency of Mobile Clients in Cloud Computing," in *Proceedings of Usenix HotCloud*, June 2010.

[29] E. Jung, Y. Wang, L. Prilepov, F. Maker, X. Liu, and V. Akella, "User-Profile-Driven Collaborative Bandwidth Sharing on Mobile Phones," in *Proceedings of ACM Mobile Cloud Computing and Services (MCS)*, June 2010.

[30] A. Shieh, S. Kandula, A. Greenberg, and C. Kim, "Seawall: Performance Isolation for Cloud Datacenter Networks," in *Proceedings of Usenix HotCloud*, June 2010.

[31] S.-W. Seong, J. Seo, M. Nasielski, D. Sengupta, S. Hangal, S. K. Teh, R. Chu, B. Dodson, and M. Lam, "PrPl: A Decentralized Social Networking Infrastructure," in *Proceedings of ACM Mobile Cloud Computing and Services (MCS)*, June 2010.

[32] Y. Xiao, P. Hui, P. Savolainen, and A. Yia-Jaaski, "CasCap: Cloud-assisted Context-aware Power Management for Mobile Devices," in *Proceedings of ACM Mobile Cloud Computing and Services (MCS)*, June 2011.

[33] P. Gilbert, B.-G. Chun, L. P. Cox, and J. Jung, "Automated Security Validation of Mobile Apps at App Markets," in *Proceedings of ACM Workshop on Mobile Cloud Computing*, June 2011.

[34] G Bruce Berriman, Ewa Deelman, Paul Groth, and Gideon Juve, "The application of cloud computing to the creation of image mosaics and management of their provenance," *SPIE Conference 7740 Software and Cyberinfrastructure for Astronomy*, 2010.